VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Komprese grafů

Graph Compression

Ondřej Szkandera

Diplomová práce

Vedoucí práce: doc. Mgr. Jiří Dvorský, Ph.D.

Ostrava, 2022

VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Ondřej Szkandera**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Komprese grafů
Graph Compression

Jazyk vypracování: čeština

Zásady pro vypracování:

Náplní diplomové práce je studium možností komprese rozsáhlých grafů. Cílem práce je implementace prototypu knihovny pro kompresi grafů a rozbor dosažených experimentálních výsledků.

Práce bude obsahovat:
1. Přehled současného stavu poznání v dané oblasti - dostupné algoritmy, knihovny a testovací grafy.
2. Návrh vlastní kompresní metody, případně adaptace již existujících metod.
3. Implementace knihovny.
4. Experimenty a jejich vyhodnocení.
5. Závěr.

Seznam doporučené odborné literatury:

[1] U Kang. Mining tera-scale graphs: Theory, engineering and discoveries. Carnegie Mellon University, 2012.
[2] Amit Chavan. An Introduction To Graph Compression Techniques For In-memory graph Computation. University of Maryland, https://www.cs.umd.edu/sites/default/files/scholarly_papers/Chavan.pdf

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **doc. Mgr. Jiří Dvorský, Ph.D.**

Datum zadání: 01.09.2021

Datum odevzdání: 30.04.2022

doc. Ing. Petr Gajdoš, Ph.D.
vedoucí katedry

prof. Ing. Jan Platoš, Ph.D.
děkan fakulty

Abstrakt

Diplomová práce se zabývá kompresí rozsáhlých grafů se snahou o zachování možnosti dekomprese požadované části bez nutnosti dekomprese celého grafu. V první části práce je shrnuta teorie spojena s touto problematikou včetně popisu několika algoritmů řešící problém komprese grafu. Druhá část práce je zaměřena na praktické problémy komprese grafů obsahující implementaci vybraného algoritmu s ohledem na efektivitu implementace. Závěr práce obsahuje zhodnocení implementace vybraného algoritmů aplikací tohoto algoritmu na několika vybraných grafech.

Klíčová slova

komprese grafu, algoritmy komprese

Abstract

The thesis deals with compression of vast graphs in an effort to preserve possibility of reverse decompression of required part without necessity of decompression the whole graph. In the first part of this thesis the theory of this issue is summarized including description of several selected algorithms solving graph compression problem. The second part of this thesis is focused on practical problems of graph compression including implementation of selected algorithm with respect to effectivity of implementation. Conclusion of this thesis contains evaluation of implementation of selected algortihm by application of this algorithm to several selected graphs.

Keywords

graph compression, compression algorithms

Poděkování

Rád bych na tomto místě poděkoval panu doc. Mgr. Jiří Dvorský, Ph.D., za odborné rady a vedení při tvorbě této práce a své rodině za podporu a trpělivost.

Obsah

Seznam použitých zkratek a symbolů

$G(V, E)$	–	graf G s množinou vrcholů V a množinou hran E
$deg(v)$	–	stupeň vrcholu v v grafu G
$E(G)$	–	množina hran grafu G
$V(G)$	–	množina vrcholů v grafu G
$\|V(G)\|$	–	počet vrcholů v grafu G
$\|E(G)\|$	–	počet hran v grafu G
$V_G(v)$	–	množina sousedů vrcholu v v grafu G
$\|V_G(v)\|$	–	počet sousedů vrcholu v v grafu G
$V_G(v)_i$	–	i-tý soused vrcholu V, pro $i = 0, 1, \ldots, \|V_G(v)\| - 1$
G_v	–	vrchol v v grafu G
$avg(G)$	–	průměrný stupeň vrcholu v grafu G
$C{=}(N, Ex, Ri, Rl)$	–	komprimovaný graf kde: N je množina vrcholů, Ex je množina uspořádaných dvojic představující extra vrcholy, Ri je množina uspořádaných dvojic vyjadřující vztah reference a Rl je množina uspořádaných dvojic reprezentující referenční seznam daného vrcholu
$N_C(x)_{Ex}$	–	uspořádaná množina extra sousedů vrcholu x v komprimovaném grafu C
$N_C(x)_{Rl}$	–	uspořádaná k-tice, představující referenční seznam pro vrchol x v komprimovaném grafu C
$N_C(x)_{Ri}$	–	množina vrcholů, která je referencována z vrcholu x
$\|N_C(x)_{Rl}\|$	–	velikost referenčního listu
$N_C(x)_{Rl_i}$	–	i-tá pozice v uspořádané k-tici referenčního listu
PSK	–	Průměrný shlukovací koeficient

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Na nejobecnější rovině můžeme v souvislosti s kompresí narazit na dva základní pojmy. Jedná se o *ztrátovou* a *bezeztrátovou* kompresi. V případě ztrátové komprese dochází pro dosažení vyššího kompresního poměru ke ztrátě určitého množství dat. Tato technika se používá například při kompresi obrázků. V případě komprese grafů vyžadujeme, aby měla konečná struktura stejnou výpovědní hodnotu jako struktura původní. Z tohoto důvodu se při kompresi grafů využívá principu bezeztrátové komprese.

Existuje několik způsobů a technik, které se při bezeztrátové kompresi využívají. Tyto techniky komprese můžeme rozdělit do dvou základních kategorií. První kategorií je *sumarizace*. [1] Princip sumarizace spočívá v nalezení opakujících se vzorů v datech, které jsou následně sjednoceny v jeden celek. Ve své podstatě se jedná o velice jednoduchý koncept, za kterým se ovšem skrývá několik úskalí. Tyto úskalí budou podrobněji rozepsány v dalších kapitolách.

Další technikou využívanou při kompresi je *kódování*. Způsobů kódování existuje celá řada. Hlavní myšlenkou kódování je „změna“ struktury dat do podoby, kterou jsme schopni reprezentovat s využitím menšího množství paměti. Velmi důležitým principem při kódování je to, že konečná podoba datové struktury má stejnou výpovědní hodnotu jako původní datová struktura, což je jeden z aspektů vycházející z principů bezeztrátové komprese. Pro dosažení co největšího kompresního poměru je dobré obě techniky kombinovat.

V souvislosti s pojmem bezeztrátové komprese se mnoha lidem často vybaví nástroje jako je ZIP apod. V rámci této práce budou uvažovány pouze takové techniky, pomocí kterých je možné provést dekompresi pouze určité části grafu, bez nutnosti dekomprese celého grafu. Rovněž budou uvažovány pouze takové grafy, které se se svou velikostí vejdou do hlavní paměti počítače, a které jsou „kompletní“ ve smyslu celistvosti grafu. Nejsou tedy uvažovány takové grafy, které by bylo nutné nějakým způsobem analyzovat po částech z důvodu příliš velkého objemu nebo nutnosti získávat graf po částech.

1.1 Definice grafu

V rámci této práce, budeme uvažovat graf jako model, který reprezentuje objekty a vztahy mezi nimi. Graf G je pak definován jako uspořádaná dvojice $G = (V, E)$, kde V je neprázdná množina vrcholů a $E \subseteq \{\{u, v\} | u, v \in V\}$ označuje množinu dvouprvkových podmnožin množiny V. Prvky množiny E jsou pak označeny jako hrany. Takto definovaný graf neumožňuje existenci hrany se stejným počátečním a koncovým vrcholem. Tento typ hran se nazývá *smyčka*. Abychom v grafu umožnili tvorbu smyček, museli bychom množinu hran vyjádřit jako $E' \subseteq E(G) \cup V(G)$, kde $E(G)$ označuje množinu hran grafu G a $V(G)$ označuje množinu vrcholů grafu G. V tomto případě bude tedy smyčka vyjádřena jednoprvkovou množinou.

Předchozí definice rovněž neumožňuje rozlišovat mezi hrany uv a vu, i přesto, že v některých případech má smysl o tomto rozlišení hran uvažovat. Abychom byli schopni tyto hrany rozlišovat, budeme původní definici grafu používat pro graf označený jako *neorientovaný graf* a graf, který rozlišuje pořadí vrcholů hrany budeme označovat jako *orientovaný graf*. Definice tohoto grafu je pak $D = (V, A)$, kde V je neprázdná množina vrcholů a A je libovolná podmnožina kartézského součinu $V \times V$ (množina uspořádaných dvojic prvků z V). Prvky z množiny A jsou označovány jako *orientované hrany*. Pokud budeme mít nějaké prvky $u \in A$ a $v \in A$, pak orientovanou hranu označíme jako uv, kde u je označován jako počáteční vrchol a v je označován jako koncový vrchol [2].

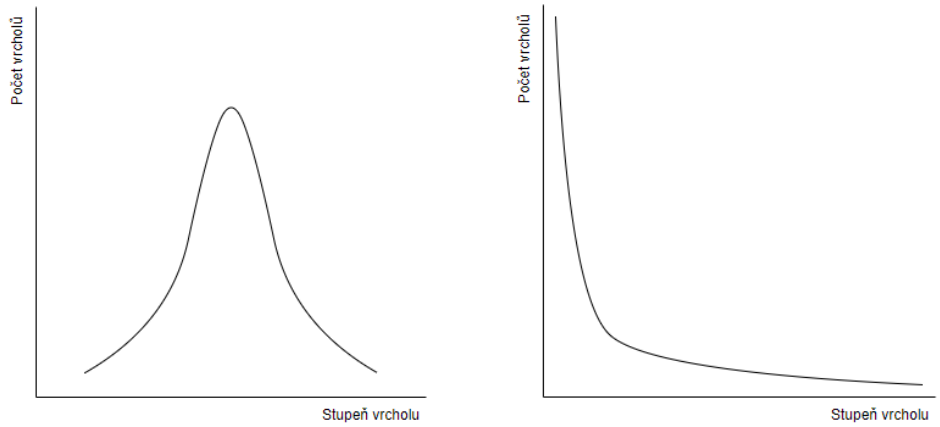
1.1.1 Webový graf

Webový graf je graf, ve kterém vrcholy reprezentují webové stránky a hrany reprezentují odkazy mezi těmito stránkami. Jedná se o orientovaný graf, ve kterém je hrana mezi vrcholy x a y v případě, kdy webová stránka x obsahuje odkaz na webovou stránku y. Webové grafy jsou rovněž typickým zástupcem bezškálových sítí s distribučním rozdělením stupňů vrcholů odpovídající mocninnému zákonu, což v důsledku znamená to, že v grafu existuje menší množství vrcholů s vysokým stupněm a mnoho vrcholů s nízkým stupněm, na rozdíl například od náhodného grafu, který má typicky Gaussovo (normální) rozdělení stupňů vrcholů. Rozdíl mezi mocninným a Gaussovým rozdělením je pak možné vidět na obrázku 1.1.

1.2 Související práce

Vzhledem k tomu, že se využití grafů v poslední době podstatně rozšířilo, vzniklo i k tomuto tématu mnoho článků, které mimo jiné také pojednávají o možnostech různých způsobů komprese.

Jedním z nich je práce od autorů Paolo Boldi a Sebastiano Vigna s názvem *The WebGraph Framework 1: Compression techniques* [3]. Článek pojednává o možnostech komprese především webových grafů. Techniky popsané v tomto článku vykazovaly po nějakou dobu nejlepší kompresní výkon, nicméně počítají s předpokladem, že jsou jednotlivé vrcholy seřazeny podle lexikografického



(a) Gaussovo rozdělení (b) Mocninné rozdělení

Obrázek 1.1: Rozdíl mezi Gaussovým a mocninným rozdělením

uspořádání názvů URL adres. Tato vlastnost neplatí pro obecné grafy a z toho důvodu se tyto techniky využívají především pro kompresi webových grafů.

Na podobné téma se zaměřují také autoři v článku *Graph Compression by BFS* [4]. Na rozdíl od předchozí práce je zde pro uspořádání vrcholů použit algoritmus BFS (průchod grafu do šířky). Díky této modifikaci lze, zde popsaný kompresní algoritmus, aplikovat na obecné grafy s poměrně vysokým kompresním poměrem.

Posledním zde zmíněným článkem zabývajícím se kompresi především webových grafů je článek od autorů Gregory Buehrer a Kumar Chellapilla s názvem *A scalable pattern mining approach to web graph compression with communities* [5]. Tento článek popisuje techniku založenou na nalezení hustých opakujících se bipartitních podgrafů. Pro tyto podgrafy jsou následně vytvořený virtuální vrcholy, které redukují počet vazeb.

Článek od autorů Sebastian Maneth a Fabian Peternek s názvem *Compressing graphs by grammars* [6] se již nezaměřuje pouze na webové grafy, ale popisuje algoritmus vycházející z bezkontextových gramatik.

V článku *An effective graph summarization and compression technique for a large-scaled graph* se autoři zaměřují na definování rozdílů mezi sumarizací a kompresí. Jedním z cílů práce je také snaha o nalezení struktury grafu s nejlepším kompresním poměrem.

Další příklad techniky komprese grafu je popsán v práci *Graph Compression Using Pattern Matching Techniques* [7]. Zde autor popisuje kompresní algoritmus vycházející z datové struktury, která se nazývá matice sousednosti a nalezení specifických vzoru v datech na základě předem definovaných vzorů.

S lehce odlišným konceptem přicházejí autoři Khan Kifayat Ullah, Nawaz Waqas a Lee Young-Koo v práci s názvem *Lossless graph summarization using dense subgraphs discovery* [8]. Tato práce

se zabývá kompresí grafu především z hlediska sumarizace grafu na základě vrcholů s vysokým stupněm interakce s ostatními vrcholy.

S podobným konceptem, jako je popsán v předchozím odstavci, přichází také práce *Graph compression* [9] od autora Fang Zhou. Zde je komprese postavena na principu shlukování hran, resp. vrcholů do tzv. super-hran resp. super-vrcholů.

Kapitola 2

Základní způsoby komprese grafů

Webové grafy patří, společně se sociálními grafy, mezi nejrozšířenější typy grafů. Díky přirozeným vlastnostem tohoto typu grafů je možné využít několik způsobu komprese grafů, které vykazují vyšší kompresní poměr v porovnání s jinými typy kompresních algoritmů určených pro obecné grafy. Tyto techniky jsou popsány v článku [3]. Samozřejmě jsou techniky popsané níže aplikovatelné na téměř jakýkoliv graf. Nicméně pro obecný graf nelze zaručit hodnotu kompresního poměru jako právě pro webové grafy. S tímto také souvisejí výše zmíněné vlastnosti, které budou stručně popsaný níže. Zmínku o horším kompresním poměru nad obecnými grafy lze také najít v článku [6].

Budeme tedy předpokládat, že je graf v paměti reprezentován pomoci seznamu sousedů a každý vrchol grafu má pro identifikaci přiřazené jedinečné nezáporné celé číslo. Pro lepší představu je tato struktura znázorněna v tabulce 2.1. V prvním sloupci tabulky je vyznačen samotný vrchol a druhý sloupec tabulky znázorňuje seznam sousedů (tedy seznam vrcholů incidentní s daným vrcholem).

Vrchol	**Incidentní vrcholy (sousedé)**
...	...
21	20, 21, 22, 23, 24, 25, 50, 75, 120
22	20, 21, 22, 70, 91
23	
24	12, 22, 25, 150
...	...

Tabulka 2.1: Základní podoba reprezentace grafu pomocí seznamu sousedů.

Je dobré si všimnout, že jsou hodnoty v sloupci se seznamem sousedů seřazeny vzestupně. V případě využití principu sumarizace se snažíme v datech najít jisté opakující se vzory, které jsou následně reprezentovány způsobem vyžadující menší množství paměti, než původní reprezentace. Seřazení vrcholů je dobrým způsobem pro jednoduché nalezení těchto vzorů a to hlavně z hlediska efektivity vyhledávání, kdy při zjišťování toho, zda je soused s vrcholu x zároveň sousedem vrcholu y, můžeme využít techniku půlení intervalů, která má časovou složitost $O(\log |V_G(y)|)$.

2.1 Reprezentace pomocí mezer

První technikou komprese grafu, kterou lze využít je reprezentace seznamu sousedů pomocí mezer. Tato technika je také často označována anglickým pojmem **Delta encoding** neboli rozdílové kódování.

Princip tohoto typu kódování spočívá v tom, že místo abychom reprezentovali sousedy vrcholu absolutní hodnotou identifikátoru souseda, použijeme relativní (odvozenou) hodnotu. Tato hodnota pak typicky v paměti zabírá méně místa než původní hodnota reprezentující souseda.

V tabulce 2.2 je vidět příklad toho, jak lze touto technikou zakódovat předchozí „naivní“ reprezentaci grafu z tabulky 2.1.

Vrchol	**Incidentní vrcholy (sousedé)**
...	...
21	1, 0, 0, 0, 0, 0, 24, 24, 44
22	2, 0, 0, 47, 20
23	
24	12, 9, 2, 124
...	...

Tabulka 2.2: Ukázka reprezentace grafu pomocí mezer

Mějme tedy množinu $V_G(v)$ představující všechny sousedy vrcholu v. Relativní hodnoty pro reprezentaci sousedů pak můžeme získat pomocí vzorce $x = (V_G(v)_i - V_G(v)_{i-1}) - 1$ kde $i > 1$. Pozorný čtenář si může povšimnou, že tímto způsobem lze získat relativní hodnotu všech sousedů kromě prvního a současně také to, že všechny výsledné hodnoty budou kladné. Budeme tedy chtít zaručit to, aby se i pro prvního souseda vypočítala vždy kladná hodnota a proto pro získání hodnoty prvního souseda, od které se bude následně odvíjet každá následující hodnota, se použijí dva vzorce. První vzorec $x_0 = G_v - V_G(v)_0$ se použije v případě, kdy bude výsledná hodnota kladná. V případě, že by výsledná hodnota byla záporná, použije se vzorec $x_0 = 2|G_v - V_G(v)_0| - 1$, který zaručí právě mapování na kladné čísla.

Tento způsob výpočtu je zvolen z toho důvodu, abychom se vyhnuli používání záporných čísel při reprezentaci sousedů. Díky tomu lze pak využít bez-znaménkové datové typy.

Jedním ze zásadních problému, který tento přístup vykazuje vychází ze dvou vlastností. První vlastností je *podobnost*. Můžeme říct, že webové stránky, které jsou sobě blízko (z pohledu lexikografického uspořádání) mají mnoho společných sousedů. Tato vlastnost je daná především tím, že většina webových stránek obsahuje mnoho odkazů odkazující zpět na danou stránku, čemuž odpovídá druhá vlastnost *lokalita*.

Tyto dva zmíněné principy nelze aplikovat na obecný graf. Z tohoto důvodu se způsob komprese stejně jako způsoby v kapitolách 2.2 a 2.3 využívají především pro webové grafy, pro které jsou tyto dvě vlastnosti typické.

2.2 Referenční komprese

Referenční komprese je další z technik, jak lze poměrně jednoduše zvýšit kompresní poměr především (ne však nutně) webových grafů. Princip této techniky spočívá v tom, že místo toho, abychom reprezentovali každý vrchol přímo, můžeme využít *podobnosti* mezi dvěma nebo více vrcholy. Vrchol pak můžeme reprezentovat jako modifikaci jiného vrcholu s informací toho, na který vrchol se daný vrchol odkazuje.

K reprezentaci dat pomocí této techniky můžeme přistoupit dvěma způsoby. První „naivní“ způsob předpokládá, že k tomu, aby byl vrchol G_v (dále v textu obecně označován jako *odkazující se* vrchol) schopny odkazovat na vrchol G_y (dále v textu označován *odkazovaný vrchol*), musí platit že $deg(G_v) \leq deg(G_y)$ a zároveň $V_G(v) \subseteq V_G(y)$. *Slovně vyjádřeno musí platit to, že stupeň vrcholu* $V_G(v)$ *je menší nebo roven stupni vrcholu* $V_G(y)$ *a zároveň vrchol* $V_G(v)$ *je incidentní se všemi sousedy vrcholu* $V_G(y)$. Pokud bude toto pravidlo dodrženo, je možné rozšířit datovou strukturu o příznak toho, na který z vrcholů daný vrchol odkazuje. Pokud v tomto případě nastane situace, kdy je stupeň odkazovaného vrcholu G_y větší, než stupeň odkazujícího se vrcholu G_v, tedy kdy platí $deg(G_v) < deg(G_y)$, znamená to, že je vrchol G_y incidentní s vrcholy, se kterými vrchol G_v incidentní není. V takovém případě je nutné tyto vrcholy vyloučit z původního seznamu sousedů a umístit je do do jiného seznamu, aby bylo jasné, kteří sousedé odkazovaného vrcholu jsou zároveň sousedy i odkazujícího se vrcholu.

Tento způsob je poměrně intuitivní, ale vykazuje několik zásadních nedostatků. Jedním z nejzásadnějších nedostatků je fakt, že pokud by byl odkazující se vrchol incidentní třeba pouze s jedním vrcholem, se kterým odkazovaný vrchol incidentní není, pak není možné provést referenční kompresi mezi těmito dvěma vrcholy. Druhým velkým nedostatkem je fakt, že pokud vrchol G_v odkazuje na vrchol G_y, pak na vrchol G_y může odkazovat pouze vrchol G_v a nebo vrchol, který je s vrcholem G_v z hlediska sousedů zcela identický (tj. takový vrchol G_x, který má stejnou množinu sousedů jako vrchol G_v). Před tím, než začne vrchol G_v odkazovat na vrchol G_y, musí být množina sousedů vrcholu G_y upravena tak, aby obsahovala pouze prvky množiny sousedů G_v (zbylé prvky množiny sousedů G_y jsou pak uloženy separátně od této množiny). Jakmile bychom chtěli na tento vrchol odkazovat z jiného vrcholu, který není identický s vrcholem G_v, museli bychom opět upravit původní množinu sousedů vrcholu G_y, čímž bychom ovšem narušili integritu struktury vytvořené pro odkaz s vrcholem G_v.

O něco sofistikovanější způsob, řešící výše zmíněné nedostatky, již umožňuje to, aby byl odkazující se vrchol incidentní i s takovými vrcholy, se kterými odkazovaný vrchol incidentní není. K tomu, abychom dokázali rozlišit, kteří sousedé odkazujícího se vrcholu jsou zároveň i sousedy odkazovaného vrcholu můžeme využít referenční seznam (angl. *reference list*) [3]. Referenční seznam si můžeme představit jako pole bitů, kdy na každé pozici v poli je hodnota 1 v případě, kdy je soused odkazujícího se vrcholu na dané pozici zároveň sousedem i odkazovaného vrcholu. Tímto lze docílit snadnějšího provázání odkazů na jednotlivé vrcholy, čímž je ve většině případů docíleno zvý-

šení kompresního poměru (i přesto, že je nutné udržovat více „nadbytečných“ informací). Základní myšlenka této techniky je znázorněna v tabulce 2.3.

Vrchol	Reference	Referenční seznam	Extra vrcholy
...	...	...	...
21	-		20, 21, 22, 23, 24, 25, 50, 75, 120
22	21	111000000	75, 91
23	-	-	
24	21	001001000	12, 150
...	...	...	...

Tabulka 2.3: Ukázka referenční komprese grafu

2.3 Rozdílová referenční komprese

Alternativním přístupem reprezentace referenčního seznamu je využití *intervalu* k popisu náležících vrcholů. Princip řešení zůstává stejný jako v předchozím případě. Jediný rozdíl je ten, že místo toho, abychom použili pole bitů pro získáni informace o tom, zda odpovídající soused odkazovaného vrcholu náleží i odkazujícímu se vrcholu, použijeme celá čísla udávající interval vrcholů sousedící s daným vrcholem.

Výhodu přináší tato metoda ve srovnání s předchozí metodou hlavně v případech, kdy je množina sousedů nějakého vrcholu poměrně velká. V případě původní reprezentace pomocí referenčního seznamu je nutné pro každého souseda odkazovaného vrcholu udržovat informaci o to, zda je soused incidentní rovněž i s odkazujícím se vrcholem. Velikost referenčního seznamu je tedy rovna stupni odkazovaného vrcholu. Pokud je stupeň odkazovaného vrcholu příliš velký, může nastat situace, kdy je pro reprezentaci referenčního seznamu spotřebováno takové množství paměti, které může přesáhnout množství paměti ušetřené díky vytvoření odkazu mezi dvěma vrcholy. Reprezentace referenčního seznamu pomocí intervalu ve velké míře snižuje délku tohoto seznamu. Pro lepší představu je v tabulce 2.4 ukázka toho, jak by mohly být data pomocí této metody reprezentovány.

Získání hodnot referenčního bloku si můžeme představit tak, že vezmeme hodnotu referenčního listu (pole bitů) a pro každou sekvenci 1 a 0 specifikujeme délku (počet po sobě jdoucích 1 nebo 0) bloků. Pro větší efektivitu můžeme navíc odebrat poslední hodnotu referenčního bloku, kterou lze z předchozí hodnoty vždy dopočítat. Rovněž ovšem musíme uchovávat informaci o tom, jestli je na začátku bloku sekvence 1 nebo sekvence 0. Rozdíl můžeme vidět v tabulce 2.3, kde v případě vrcholu 22 referenční list začíná sekvencí 1 a vrchol 24 začíná sekvencí 0. Bez této informace nejsme schopni provést dekompresi. V tabulce 2.4 je sloupec s touto hodnotou označen jako *Iniciátor*.

Důležitou podmínkou pro to, abychom mohli tuto metodu použít je to, aby byl seznam sousedů v každém vrcholu seřazen. Rovněž je potřeba si uvědomit, že tento způsob reprezentace je vhodný pouze v případech, kdy jsou jednotlivé bloky velmi dlouhé a celkový počet bloků je malý. Vezměme

Vrchol	Reference	Referenční bloky	Iniciátor	Extra vrcholy
...	...	...	...	...
21	-	-	-	20, 21, 22, 23, 24, 25, 50, 75, 120
22	21	3	1	75, 91
23	-	-	-	-
24	21	2, 1, 2, 1	0	12, 150
...	...	...	...	

Tabulka 2.4: Ukázka referenční komprese grafu s využitím bloků

v úvahu například situaci, kdy budeme chtít, aby vrchol G_v odkazoval na vrchol G_y. Stupeň vrcholu G_y je 1024 (tj. vrchol G_y má 1024 sousedů) a algoritmicky bylo zjištěno, že celkový počet bloků potřebných pro reprezentaci odkazu je 20. Dále budeme uvažovat, že hodnoty jednotlivých bloků jsou reprezentovaný celým číslem, a že reprezentace celého čísla v paměti vyžaduje 64 bitů. V případě referenčního seznamu je nutné vyčlenit na jeho reprezentaci 1024 bitů. Seznam referenčních bloku pak vyžaduje $20 \times 64 = 1280$ bitů. Z tohoto jednoduchého výpočtu je viditelné, že pokud bude seznam referenčních bloků obsahovat větší množství položek, může nastat situace, kdy velikost paměti potřebné pro jeho reprezentaci přesáhne množství paměti potřebné pro reprezentaci referenčního seznamu se stejnou výpovědní hodnotou. Tato problematika je opět částečně provázána s vlastnostmi webových grafů, které jsou pro tuto reprezentaci vhodnější, než jiné grafy. Dalo by se tedy říct, že rozdílová referenční komprese je z obecného hlediska méně efektivní než referenční komprese s využitím referenčního seznamu.

2.4 Řetězení referencí

Pokud použijeme pro kompresi jednu z předchozích metod nebo jejich kombinaci, pak musíme myslet na několik faktorů ovlivňující efektivitu komprese. Jedním z těchto faktorů je například *velikost referenčního okna* W. Tato hodnota shora ohraničuje počet možných odkazů na sousedy odkazovaného vrcholu. Pro počet možných sousedů, na které je (z odkazujícího vrcholu) možné odkázat (označíme r) tedy platí $0 < r < W$.

Dalším důležitým parametrem je *referenční řetězec vrcholů* (angl. *reference chain of node*). Jedná se o parametr, který shora ohraničuje počet rekurzivních zanoření odkazů jednoho vrcholu. Rekurzivní zanoření vrcholů si můžeme představit tak, že vrchol X má odkaz na vrchol Y, vrchol Y má odkaz na vrchol Z atd. V tomto našem případě nastává ten problém, že pokud bychom chtěli získat všechny sousedy vrcholu X, museli bychom dekomprimovat jak vrchol Y, tak i vrchol Z a všechny další vrcholy v referenčním řetězci. Pokud by tedy toto zanoření nebylo omezené, mohla by (v nejhorším případě) nastat situace, kdy bychom byli nuceni pro získání seznamu sousedů nějakého vrcholu dekomprimovat celý graf.

Ponecháme-li tento parametr neomezený, zvyšujeme efektivitu komprese (statické), ale zároveň snižujeme efektivitu komprese při dotazech (dynamické).

Řetězení referencí není v některých případech takový problém jako v jiných. Například pokud víme, že budeme grafem procházet pouze sekvenčně, pak je tento problém zanedbatelný. Na druhou stranu, pokud víme, že budeme přistupovat velmi často k jednotlivým vrcholům, pak musíme hodnotu tohoto parametru volit obezřetně.

Kapitola 3

Další způsoby komprese grafů

V rámci této části budou pouze stručně popsány další algoritmy, které již nejsou zaměřeny pouze na kompresi webových grafů, ale dívají se na graf jako na obecnou strukturu, která předem nepředpokládá s žádnými vlastnostmi.

3.1 Komprese grafů pomoci bezkontextových gramatik

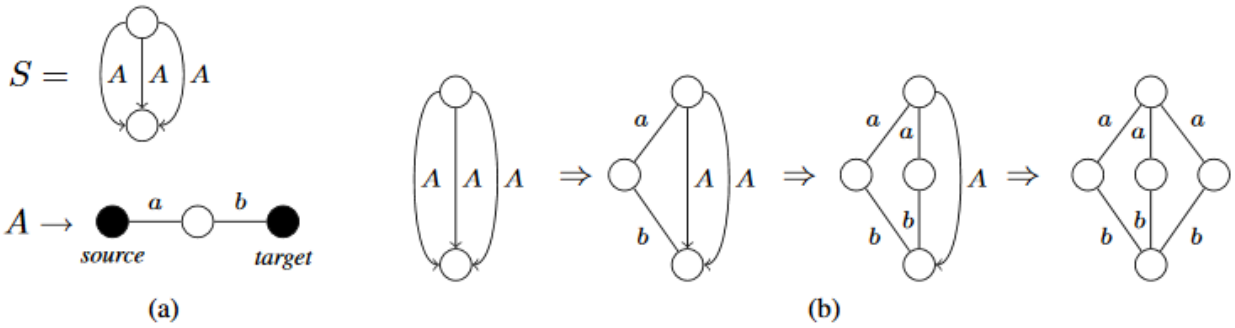
Kompresí grafů pomocí bezkontextových gramatik se zabývají autoři v práci [6]. Celá myšlenka, jak už z názvu vypovídá, vychází z nápadu převést redundantní strukturu dat do podoby bezkontextové gramatiky tak, aby se co nejvíce zabránilo opakujícím se strukturám v grafu. Technika vychází z kompresního schématu s názvem *RePair* [10].

Myšlenkou *RePair* algoritmu je opakované nahrazování nejfrekventovanějších digramů (pokud bychom předpokládali nějaký řetězec znaků, pak je digramem myšlena dvojice sousedních písmen) novým neterminálním symbolem do té doby, dokud se daný digram nebude vyskytovat v textu maximálně jednou. Komprese pomocí gramatik tedy přebírá tuto myšlenku s tím rozdílem, že za digram považuje dvojici propojených hyper-vrcholů, kde jeden hyper-vrchol může obsahovat více jak 2 sousední vrcholy.

Pro lepší pochopení problematiky je na obrázku 3.1 [6] znázorněn příklad převedení části grafu na neterminál.

Na obrázku 3.1 lze vidět, že byl vytvořen nový neterminál A, který reprezentuje vrchol, z něhož vycházejí 2 hrany do stejných dvou vrcholů. V obrázku vlevo na hoře můžeme vidět reprezentaci grafu s použitím neterminálu a vpravo pak to, jak by graf vypadal, kdybychom graf vyjádřili v původní reprezentaci bez neterminálů. Kompresí bylo dosaženo jak snížením počtu hran (z 6 na 3), tak i snížením počtu vrcholů (z 5 na 2).

Hlavním problémem této techniky je nalezení maximálního počtu co největších nepřekrývajících se digramů. S tímto částečně také souvisí další problém, a to nalezení co nejmenší gramatiky pro daný řetězec. Jedná se však o typ NP-problému [11].



Obrázek 3.1: Převod části grafu na neterminál [6]

3.2 Komprese grafů založena na porovnávání vzorů

Další z technik pro kompresi grafů je popsaná v článku [7]. Tato technika je založena na identifikaci vzorů v grafu reprezentovaného pomocí matice sousednosti a následné transformace všech částí grafů odpovídajících daným vzorům za tzv. značky, které typicky vyžadují pro reprezentaci menší množství paměti.

Algoritmus je navržen spíše pro grafy s velkým počtem vrcholů, ale naopak s malým počtem hran, kde je výsledná matice sousednosti řídká. Nicméně je samozřejmě možné aplikovat tento algoritmus na jakýkoliv obecný graf. Pro hustější grafy ovšem roste počet vzorů, kterým může potencionálně odpovídat datový řetězec a z toho důvodu klesá efektivita algoritmu.

Hlavní princip algoritmu spočívá v rozdělení všech řádků na části o velikosti rovnající se velikosti vzorů. Tyto části jsou následně spárovány s vyhovujícím vzorem. Každá úspěšně spárovaná část je nahrazena identifikátorem vzoru, přičemž hodnota 1 na začátku znamená, že následné bity reprezentují identifikátor. Ostatní části jsou předsazeny 0 a vloženy v čisté podobě.

Nevýhodou algoritmu je fakt, že se všechny části, které neodpovídají žádnému vzoru uchovávají nekomprimovaná. Z tohoto vyplývá to, že efektivita komprese bude záležet (kromě samotných dat) i na volbě velikostí vzorů. Další nevýhodou algoritmu je fakt, že pracuje s grafem reprezentovaným pomocí matice sousednosti, která je sama o sobě z hlediska paměťových nákladů poměrně neefektivní pro uchovávání rozsáhlejších grafů.

3.3 Komprese grafu s pomocí BFS algoritmu

BFS kódování, tak jak jej nazvali autoři v článku [4], je svým způsobem pouze upravená verze algoritmů z kapitoly 2. Zatímco v kapitole 2 se na pořadí vrcholu pohlíželo z pohledu lexikografického uspořádání URL adres, tak v této práci autoři zobecňují efektivitu komprese na úroveň obecného grafu a to tím, že se již nedívají na graf z hlediska lexikografického uspořádání jeho vrcholů, ale z pohledu topologie grafu.

Pro výpočet BFS kódování je použit, jak již z názvu vypovídá, algoritmus BFS, tedy algoritmus prohledávání do šířky (angl. *Breadth-First search*). Celý algoritmus je rozdělen do dvou fází. V první fázi se pomocí algoritmu BFS projde celý graf a jednotlivé navštívené vrcholy se označí čísly v pořadí, ve kterém byly vrcholy navštíveny. Tato sekvence vrcholů je v algoritmu označena jako *traversal list*. Následně jsou v první fázi komprimovány po sobě jdoucí bloky grafu o určité velikosti l, kde l je obezřetně stanovená hodnota označovaná jako úroveň komprese. Ve druhé fázi se zakóduje seznam sousedů komprimovaných bloků pomocí algoritmů popsaných v kapitole 2.

Výhodou tohoto řešení je to, že se dopředu nepočítá s žádnými speciálními vlastnostmi grafu jako tomu bylo například v kapitole 2.

Kapitola 4

Implementace

Úvodem implementační části by bylo dobré vymezit hranice a cíle výsledného algoritmu. Cílem algoritmu je vytvoření datové struktury, která umožní efektivní paměťovou reprezentaci grafu vyžadující menší množství paměti, než původní reprezentace stejného grafu s ohledem na zachování možnosti procházení grafem bez nutnosti dekomprese celého grafu. Rovněž jsou v rámci implementace uvažovány pouze takové grafy, které je možné uložit celé do paměti. S ohledem na tyto požadavky byla pro implementaci vybraná metoda, která je popsaná v kapitole 2.2 spolu s omezeními popsanými v kapitole 2.4.

4.1 Způsoby reprezentace grafu v počítači

Způsob reprezentace grafu v počítači je jednou ze základních otázek, kterou je nutné zodpovědět pro to, abychom byli schopni s grafy efektivně pracovat. Každý způsob implementace přináší určité výhody a nevýhody pro určité operace nad grafy. V rámci této kapitoly budou popsány základní způsoby, jakými lze grafy v aplikaci reprezentovat a jaké jsou výhody, popřípadě nevýhody zvoleného řešení.

V důsledku je pak způsob reprezentace grafu i samotnou otázkou komprese, kdy se v rámci kompresního algoritmu snažíme najít takový způsob reprezentace, který umožní popsat celý graf s co možná nejmenším množstvím využité paměti.

4.1.1 Reprezentace pomocí matice sousednosti

Matice sousednosti je matice o velikosti $m \times m$, kde m je počet vrcholů v grafu. Prvky matice mohou nabývat hodnot 0 nebo 1 pro neohodnocené grafy a pro ohodnocené grafy je možné na konkrétních pozicích matice specifikovat váhu hrany mezi dvěma vrcholy. V případě neorientovaného grafu je matice symetrická. Tento typ reprezentace dat se používá například v metodě komprese popsané v kapitole 3.2.

Velkou nevýhodou tohoto způsobu reprezentace je množství využité paměti. Vezměme například graf s 10^5 vrcholy. Pro jeho reprezentaci je nutné vytvořit matici o velikosti $10^5 \times 10^5$. Pokud budeme uvažovat reprezentaci grafu s neváženými hranami, pak je možné pro reprezentaci použít datový typ *boolean*. Při předpokladu, že pro reprezentaci tohoto datového typu je nutné v paměti vyčlenit 8 bitů, pak, pak pro reprezentaci celého grafu bude využito minimálně $10^5 \times 10^5 \times 8 = 8 \times 10^{10}$ bitů, což je v přepočtu nějakých 10 GB.

Vzhledem k tomu, že cílem práce je komprimace rozsáhlých grafů, kdy budeme předpokládat, že grafy budou obsahovat více jak 10^5 vrcholů, pak je pro nás tento způsob reprezentace grafu nevhodný a v této práci nebude dále uvažován.

4.1.2 Reprezentace pomocí matice incidence

S reprezentací grafu pomocí matice incidence se moc často nesetkáme i přesto že je to asi jeden z nejjednodušších způsobu jak reprezentovat *hypergraf*[1].

Matice incidence je matice o velikosti $|V(G)| \times |E(G)|$. Jinými slovy existuje pro každý vrchol řádek a pro každou hranu existuje sloupec.

To, jaký vrchol je incidentní s jakou hranou je opět naznačeno tím, že se daný průsek řádku a sloupce označí hodnotou 1 pro neohodnocený graf a hodnotou příslušné váhy hrany pro ohodnocený graf. V případě, že pracujeme s orientovaným grafem, je možné orientaci hran reprezentovat pomocí záporné hodnoty v průseku řádku a sloupce reprezentující počátek hrany a kladnou hodnotou v průseku řádku a sloupce reprezentující konec hrany.

Tento způsob reprezentace grafu je ovšem stejně nevýhodný z hlediska množství využité paměti jako reprezentace pomocí matice sousednosti. V případě velkého počtu hran opět bude matice poměrně rozsáhlá a z tohoto důvodu není ani tento způsob reprezentace grafu pro tuto práci vhodný.

4.1.3 Reprezentace pomocí seznamu sousedů

V této kapitole se již dostáváme k trochu odlišnějšímu řešení reprezentace grafu. Zatímco předchozí dvě řešení reprezentovaly graf poměrně přehledně i v případě, že bychom si jej vynesli například na papír nebo do tabulkového procesoru, tak v případě reprezentace grafu pomocí seznamu sousedů je situace poněkud jiná. Pokud bychom chtěli analyzovat nějaký menší graf zakreslený pomocí seznamu sousedů, bude to určitě daleko složitější než v případě matice sousednosti nebo incidence. Z tohoto důvodu se tento způsob reprezentace grafu používá převážně v aplikačním prostředí.

Při reprezentaci grafu pomocí seznamu sousedů se nejprve vytvoří seznam vrcholů $V(G)$ a následně se pro každý vrchol $v \in V(G)$ vytvoří seznam, obsahující všechny vrcholy, které jsou s vrcholem, v incidentní.

Hlavní výhodou tohoto způsobu reprezentace grafu je poměrně velká efektivita z hlediska využití paměti. Obě výše zmíněné reprezentace zaznamenávaly jak to, že hrana mezi vrcholy existuje, tak

[1] Speciální případ grafu, ve kterém může jedna hrana spojovat libovolný počet vrcholů.

i to, že hrana mezi vrcholy neexistuje. Tento přístup je výhodný z hlediska časové složitosti, kdy jsme schopni zjistit, jestli jsou dva vrcholy spolu incidentní přečtením pouze jedné hodnoty, ale z hlediska pamětí jsou tyto metody reprezentace velmi náročné. Namísto toho v případě reprezentace grafu pomoci seznamu sousedů do datové struktury ukládáme pouze informaci o existenci hrany a nikoli o její neexistenci. Kvůli tomu nejsme schopni identifikovat incidenci mezi vrcholy tak efektivně jako v případě reprezentace pomocí matice sousednosti nebo matice incidence, nicméně v případě rozsáhlých grafu docílíme touto reprezentací daleko kompaktnější struktury. Z tohoto důvodu je při implementaci uvažován právě tento způsob reprezentace.

4.2 Zdroj dat a jejich anonymizace

Zdroje dat můžou často přicházet v nejrůznějších podobách. Velmi často se setkáme se zdroji, které jsou distribuovány v rámci jednoho souboru tak, že na každém řádku je dvojice identifikátorů vrcholů. Každý řádek tedy reprezentuje jednu hranu a to tak, že první identifikátor reprezentuje zdrojový vrchol hrany a druhý identifikátor reprezentuje cílový vrchol hrany. K těmto dvěma informacím mohou být přidány ještě další informace například o váze hrany a podobně.

Identifikátory vrcholu jsou velmi často celá čísla. Nicméně se můžeme setkat i s případy, kdy je jako identifikátor použita například URL adresa. V takových případech je ovšem neefektivní reprezentovat vrchol jeho identifikátorem. Při reprezentaci grafu se mohou identifikátory vrcholů vyskytovat na více místech. Například v případě reprezentace grafu pomocí seznamu sousedů se identifikátor vyskytuje jak v roli identifikace samotného vrcholu, tak i v jednotlivých seznamech pro identifikaci souseda. Pokud tedy jako identifikátor použijeme řetězec namísto například celého čísla, můžeme si být jistí, že výsledná reprezentace grafu bude v paměti zabírat daleko více místa.

Z tohoto důvodu bude v rámci implementace použito přemapováni všech identifikátorů vrcholů na nezáporné celočíselné hodnoty. Tímto dosáhneme snížení paměťových nároků v případech, kdy jsou vrcholy reprezentovaný jinými identifikátory, než jsou nezáporná celá čísla.

Může se zdát, že pro datové zdroje, které již vrcholy reprezentují pomocí celočíselných identifikátorů je mapováni zbytečné. Nicméně má přemapováni zdrojových dat další výhodu a tou je jejich anonymizace. Tím, že všechny původní identifikátory vrcholů přemapujeme, nebude možné zpětně, bez znalosti mapovací funkce, jednoznačně určit přesný výchozí vrchol, což může být v některých případech žádoucí.

Zde je důležité zmínit, že se nejedná o dokonalou anonymizaci dat v pravém slova smyslu. V případě jedinečného vrcholu v grafu, tj. vrcholu, která má naprosto odlišné sousedy od všech ostatních bude stále možné identifikovat tento konkrétní vrchol. Tato práce se ovšem nezabývá anonymizací dat a z tohoto důvodu si zde vystačíme s touto jednoduchou anonymizací.

4.3 Popis algoritmu komprese

Implementovaný kompresní algoritmus je postaven na principu referenční komprese popsané v kapitole 2.2 a ve výsledku odráží podobu modelu zobrazeného v tabulce 2.4. Samotný algoritmus se skládá ze tří fází, které jsou popsány v následujících kapitolách, přičemž každou fázi je možné vykonat až po ukončení všech předchozích fází. Jednotlivé fáze tak není možné zpracovávat souběžně.

4.3.1 Fáze I – vytvoření grafu podobností

V rámci první fáze se algoritmus pokouší ke každému z vrcholu najít vrchol, který je danému vrcholu nejpodobnější z hlediska počtu společných sousedů. Výstupem algoritmu je orientovaný graf, dále označován jako *graf podobností*, který obsahuje stejnou množinu vrcholů jako původní graf a hrana mezi vrcholy existuje v případě, pokud je vrchol na výstupní straně vyhodnocen jako nejpodobnější vrcholu na vstupní straně hrany. Funkce podobnosti tedy hledá pro každý vrchol x grafu G takový vrchol y, kde platí

$$\max\{|V_G(x) \cap V_G(y)|\}$$

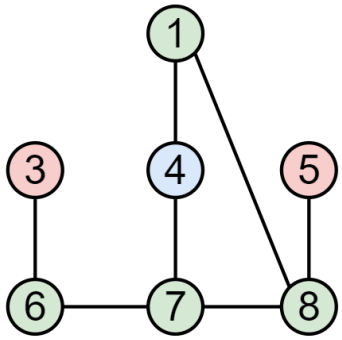
Základní princip první fáze algoritmu je znázorněn v ukázce algoritmu 1. Na vstupu funkce je původní graf a na výstupu je graf podobností. Z hlediska časové složitosti se jedná o nejnáročnější fázi celého algoritmu. Vyjádříme-li časovou složitost pomocí asymptotické notace, pak složitost této fáze algoritmu roste přibližně podle funkce $O(|V(G)|^2 \times avg(G))$.

Vzhledem k velké náročnosti výpočtu v této fázi bylo v rámci implementace provedeno několik optimalizací, které ve výsledku vedly k podstatnému snížení celkového času výpočtu. Prvním pokusem ve snaze o optimalizaci časové složitosti první fáze algoritmu bylo převedení této částí implementace do paralelní podoby. Tímto krokem bylo docíleno snížení času potřebného pro výpočet grafu podobností přibližně o polovinu. Původní problém s asymptotickou časovou složitostí však byl i po tomto kroku zachován a délka výpočtu stále dosahovala pro rozsáhlé grafy (počet vrcholu přibližně 5×10^5) řádu desítek minut. Hlavním nedostatkem tohoto „primitivního“ řešení byla nutnost porovnávat každý vrchol grafu se všemi ostatními vrcholy bez ohledu na to, jak byly vrcholy daleko od sebe (z hlediska počtu hran potřebných k jejich dosažení).

Vezměme v úvahu graf na obrázku 4.1. Graf obsahuje 8 barevně označených vrcholů. Řekněme, že se budeme snažit najít nejpodobnější vrchol pro vrchol 4 označený modrou barvou. Je jasné, že jedinými kandidáty na nejpodobnější vrcholy jsou ty vrcholy, které mají alespoň jednoho stejného souseda jako má vrchol 4. Tyto vrcholy jsou zvýrazněny zelenou barvou a v terminologii algoritmu jsou označovány jako *okolní vrcholy* (surrounding nodes). Červené vrcholy jsou pak takové vrcholy, které nemají s vrcholem 4 žádného společného souseda a tudíž není nutné je při porovnávání brát v úvahu.

Pro řešení problému s časovou složitostí v této fázi bylo využitou právě existence okolních vrcholů a faktu, že všechny ostatní vrcholy, které nepatří do okolí vrcholu jsou pro nalezení nejpodobnějšího

vrcholu irelevantní. Bylo tedy nutné do první fáze algoritmu zakomponovat logiku pro nalezení okolí pro každý vrchol.



Obrázek 4.1: Okolí z hlediska podobnosti vrcholů

Jako první varianta pro nalezení okolí vrcholu se nabízelo využití klasického přístupu pomocí prohledávání grafu například algoritmem DFS (Deep-first search)[12]. V souvislosti s tímto přístupem ovšem vznikl problém spojený s paralelismem a množstvím využité paměti, kdy při vyšší úrovni paralelizace docházelo ve velmi krátkém čase (jednotky vteřin) k extrémnímu nárůstu paměti, a to v řádech stovek až tisíců megabajtů, což ve většině případů zapříčinilo vyčerpání zdrojů běhového prostředí a pád aplikace. Řešením tohoto problému bylo využití vzoru *Iterátor* a zjednodušení algoritmu pro nalezení okolí vrcholu. Zjednodušení spočívalo v adaptaci algoritmu DFS tak, aby prohledal graf pouze do hloubky 2 od aktuálního vrcholu (uvažovat hloubku 3 a více je pro nalezení nejpodobnějšího vrcholu zbytečné, jak bylo vysvětleno v předchozí kapitole). Vzor *Iterátor* pak umožnil průchod okolními vrcholy bez nutnosti uložení seznamu okolních vrcholů do externí datové struktury a omezil množství využité paměti každého vlákna. Princip implementace nalezení okolí vrcholu pro neorientovaný graf je možné vidět v ukázce algoritmu 2. Hlavní myšlenka algoritmu zůstala zachována i pro orientovaný graf s tím rozdílem, že je nutné v původním grafu udržovat i informaci o zpětných hranách. Bez této informace by byl algoritmus z hlediska časové složitosti výpočtu velmi neefektivní.

Použití algoritmu 2 v rámci tvorby grafu podobností je možné vidět na řádku 3 v ukázce algoritmu 1. Zavedením principu prohledáváni pouze okolí vrcholů bylo docíleno podstatného snížení časové složitosti, a délka výpočtu grafu podobností se snížila na řády jednotek až desítek vteřin.

Další optimalizace první fáze jak z hlediska časového, tak i z hlediska výsledného kompresního poměru spočívala v zavedení povoleného rozptylu stupně vrcholu označovaného jako *degree dispersion*. Parametr je v ukázce 1 zohledněn na řádku 6, a udává hodnotu, jež určuje maximální odchylku stupňů vrcholů, které mohou být navzájem odkazovány. Řekněme například, že bychom nastavili

Algoritmus 1: CreateListOfSimilarNodes

```
Data: G = (V, E)
Result: G' = (V', E') orientovaný graf podobnosti
1  begin
2      for v ∈ V do
3          sv ⟵ GetSuroundingNodes(G, v)
4          currentNumberOfSameNodes ⟵ 0
5          for s ∈ sv do
6              if stupeň vrcholu je uvnitř povoleného rozptylu then
7                  numberOfSameNodes ⟵ |V_G(x) ∩ V_G(y)|
8                  if numberOfSameNodes > currentNumberOfSameNodes then
9                      sizeSave ⟵ GetTheoreticalNodeSizeSave(v, sv, numberOfSameNode)
10                     if sizeSave < 1 then
11                         currentNumberOfSameNodes ⟵ numberOfSameNodes
12                         V'_G'(v) ⟵ {s}
13                     end
14                 end
15                 if deg(v) = numberOfSameNodes then
16                     ukonči prohledávání okolí vrcholu a pokračuj dalším vrcholem
17                 end
18             end
19         end
20     end
21 end
```

Algoritmus 2: GetSurroundingNodes

```
Data: G = (V, E); v vrchol, pro který bude vyhledáno okolí
Result: množina okolních vrcholů pro vrchol v
1  begin
2      visited ⟵ ∅
3      for x ∈ V_G(v) do
4          if x ∉ visited then
5              visited ⟵ visited ∪ {x}
6          end
7          for y ∈ V_G(x) do
8              if y ∉ visited then
9                  visited ⟵ visited ∪ {y}
10             end
11         end
12     end
13     return visited
14 end
```

hodnotu $DegreeDispersion = 50$, pak to znamená, že vrchol se stupněm 100 může odkazovat pouze na vrcholy, jejichž stupeň je v intervalu $\langle 50; 150 \rangle$.

Použitím tohoto parametru omezíme velikost množiny vrcholů, kterou je nutné porovnat pro nalezení nejpodobnějšího vrcholu, což opět snižuje časovou složitost algoritmu. Hlavní přínos tohoto parametru ovšem spočívá, jak už bylo zmíněno, ve zvýšení výsledného kompresního poměru. Vzhledem k struktuře objektu (popsané v kapitole 4.3.3), který popisuje komprimovaný vrchol, není žádoucí, aby byly všechny úspěšně vyhledané podobné vrcholy zařazeny do výstupního grafu. V některých případech by mohla nastat situace, kdy algoritmus vyhodnotí jako nejpodobnějšího souseda vrcholu x, vrchol s daleko vyšším stupněm, než má vrchol x. Relativní velikost referenčního seznamu by pak mohla přesáhnout velikost, jež bychom ztratili počtem namapovaných vrcholů a velikost kompresního poměru by se snižovala.

Experimenty ukázaly, že v krajních hodnotách intervalu rozptylu povolených stupňů vrcholů stále může docházet k situaci, kdy by velikosti referenčního seznamu a bloku extra vrcholů mohla přesáhnou teoretickou velikost původní reprezentace vrcholů. Aby byly omezeny i tyto případy, byl do algoritmu přidán výpočet teoretické úspory pro každý vrchol, který by mohl být potenciálně vyhodnocen jako nejpodobnější vrchol nějakého jiného vrcholu. Výpočet teoretické úspory v případě, kdy je vrchol x uvažován jako kandidát nejpodobnějšího vrcholu k vrcholu y vychází z následujícího vzorce, kde TRS je konstanta vyjadřující teoretické množství paměti potřebné pro reprezentaci jedné reference (při implementaci uvažovaná hodnota 64), přičemž platí, že pokud je teoretická úspora větší nebo rovná 1, uvažovaný vrchol není považován za podobný a do výstupního grafu podobností se nedostane.

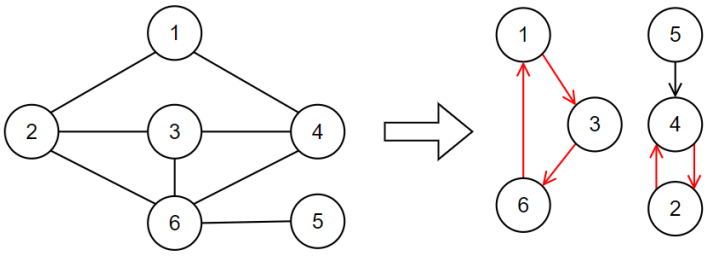
$$TS = \frac{4TRS + |V_G(x)| + \Big(|V_G(y)| - |V_G(x) \cup V_G(y)|\Big)TRS}{\Big(|V_G(y)| + 2\Big)TRS}$$

Díky použití přepočtu teoretické úspory komprese by se mohlo zdát, že ošetřovat velikost referenčního seznamu pomoci parametru *DegreeDispersion* je zbytečné. Je ovšem nutné stále brát v úvahu i to, že pro výpočet teoretické úspory je nutné nejprve získat počet společných sousedů. Tato operace může být také časově poměrně náročná, zejména pak v případě vrcholů s vysokým stupněm. Díky zachování podmínky parametru *DegreeDispersion* omezíme počet těchto operací na nezbytné minimum a zajistíme tak vyšší výkon algoritmu.

4.3.2 Fáze II – detekce a odstranění cyklů v grafu podobností

V první fázi algoritmu byl vytvořen graf podobností. Hlavním požadavkem tohoto grafu je, aby neobsahoval žádný cyklus. Pokud by výsledný graf obsahoval cyklus, mohlo by při zpětné rekonstrukci grafu dojít k zacyklení dekompresního algoritmu. Celý problém je nastíněn v ukázkovém grafu na obrázku 4.2, kde můžeme vidět, že algoritmus pro původní graf (na levé straně) vygeneroval oriento-

vaný graf podobností (na pravé straně) obsahující cykly. Pro lepší přehlednost jsou cykly v obrázku vyznačeny červenou barvou. Pokud bychom provedli kompresi bez odstranění těchto cyklů, nastal by problém pří zpětné rekonstrukci sousedů prakticky u všech vrcholů.



Obrázek 4.2: Ukázka problému se zacyklením grafu podobnosti

Vezměme v úvahu například požadavek na rekonstrukci vrcholu 5. Aby bylo možné tento vrchol zrekonstruovat, je nutné znát všechny sousedy vrcholu 4. K tomu, abychom byli schopni rekonstruovat vrchol 4, musíme zjistit všechny sousedy vrcholu 2. A právě zde nastává problém, protože pro zjištění všech sousedů vrcholu 2 potřebujeme znát všechny sousedy vrcholu 4. V algoritmu tak vzniká nekonečná smyčka, protože není možné sestavit vrchol 4 bez toho, abychom znali všechny sousedy vrcholu 2 a současně není možné sestavit vrchol 2, protože bychom museli znát všechny sousedy vrcholu 4.

Z tohoto příkladu je možné vidět, že existence cyklů v orientovaném grafu podobností je velký problém pro zpětnou rekonstrukci. I přesto, že vrchol 5 nebyl přímou součástí žádného cyklu, nebylo možné jej zpětně sestavit. Podobný problém pak můžeme pozorovat u všech vrcholů výsledného grafu podobností.

Vzhledem k paralelnímu řešení první fáze algoritmu není možné řešit detekci cyklů již při sestavování grafu podobností bez toho, abychom razantně zvýšili časovou složitost první fáze. Vzhledem k tomu, že je první fáze algoritmu nejnáročnější jak z hlediska časové tak i paměťové složitosti, je nutné zachovat maximální efektivitu sestavení grafu podobností.

Detekce a následné odstranění cyklů je proto řešena samostatně v rámci druhé fáze algoritmu, kdy již máme k dispozici kompletně sestavený graf podobností. Jádrem druhé fáze je algoritmus pro detekci cyklů v grafu. Jako referenční řešení tohoto úkolu byla částečně převzatá implementace z článku *Detekce cyklu v orientovaném grafu* [13]. Implementace tohoto algoritmu však byla navržena pouze pro detekci jednoho cyklu. V případě detekce cyklu algoritmus skončil s informací o existenci cyklu v grafu. Vzhledem k tomu, že pro účely komprese bylo nutné detekovat a současně odstranit všechny cykly v grafu, bylo nutné původní implementaci upravit do efektivnější podoby. Základní princip algoritmu je možné vidět v ukázce 3 a 4. V ukázce 3 je pseudokód, který vnitřně využívá logiku algoritmu v ukázce 4, a který zároveň obsahuje logiku pro odstranění cyklů v grafu. Rekurzivní

Algoritmus 3: DetectLoop

Data: $G = (V, E)$ graf podobností

```
1  begin
2      unvisitedNodes ⟵ V
3      recStack ⟵ ∅
4      for x ∈ unvisitedNodes do
5          unvisitedNodes ⟵ unvisitedNodes − {x}
6          isLoopDetected ⟵ DetectLoopInternal(G, x, recStack)
7          if isLoopDetected = TRUE then
8              nodeInLoop ⟵ recStack[|recStack| − 2]
9              V_G(nodeInLoop) ⟵ ∅
10             recStack ⟵ ∅
11         end
12     end
13 end
```

Algoritmus 4: DetectLoopInternal

Data: $G = (V, E)$ graf podobností; x počáteční vrchol; $recStack$ zásobník rekurce
Result: $TRUE$ pokud je detekován cyklus, jinak $FALSE$

```
1  begin
2      if recStack ∩ {x} ≠ ∅ then
3          return TRUE
4      end
5      recStack ⟵ recStack ∪ {x}
6      if V_G(x) ≠ ∅ then
7          similarNode ⟵ V_G(x) //z povahy věci obsahuje množina maximálně 1 prvek
8          isLoopDetected ⟵ DetectLoopInternal(G, similarNode, recStack)
9          if isLoopDetected = TRUE then
10             return TRUE
11         end
12     end
13     recStack ⟵ recStack − {x}
14     return FALSE
15 end
```

algoritmus 4 obsahuje logiku pro detekci cyklů v orientovaném grafu a jeho výstupem je informace o tom, zda byl cyklus detekován. V případě pozitivní detekce cyklu je pak možné detekovaný cyklus získat z proměnné v algoritmu označené jako *recStack*.

Vzhledem k tomu, že graf podobností obsahuje stejný počet vrcholů, jako původní graf, tak je nutné i při detekci cyklů brát v potaz efektivitu implementace. Z tohoto důvodu je v algoritmu množina označená jako *unvisitedNodes*, která obsahuje všechny vrcholy, které zatím nebyly zkontrolovány. V případě detekce cyklu algoritmus tento cyklus odstraní tím, že zruší vazbu mezi jedním z vrcholů tvořící cyklus a pokračuje dále v prohledáváni grafu vrcholem, který ještě nebyl zkontrolován. V případě pozitivního detekování cyklu tak není nutné se vracet na začátek grafu a začít prohledáváni od začátku, jako by to bylo nutné v případě původní implementace algoritmu z článku [13].

V průběhu druhé fáze algoritmu dochází k odstraňování hran z grafu podobností. To zapříčiňuje v určité míře snížení velikosti kompresního poměru. Z tohoto důvodu bylo při prvotní implementaci uvažováno o zefektivnění procesu ve smyslu přesunutí hrany namísto jejího odstranění. Tento přístup ovšem velmi snižoval efektivitu algoritmu z hlediska časové složitosti, a z tohoto důvodu byla následně implementována výše popsaná varianta algoritmu.

Následně se v průběhu testování algoritmu ukázalo, že se množství detekovaných cyklů pro různé grafy velmi liší. Obecně je v rámci tohoto algoritmu složité predikovat jeho chování na základě nějakých strukturálních vlastností původního grafu. V některých případech byl počet detekovaných cyklů velmi nízký i pro grafy s velkým množstvím vrcholů a hran. V jiných případech naopak bylo detekováno tolik cyklů, že byl počet referencí snížen prakticky o polovinu.

4.3.3 Fáze III – finální sestavení komprimovaného grafu

Po úspěšném odstranění všech cyklů v grafu podobnosti nastupuje 3. fáze algoritmu, což je konečné sestavení komprimovaného grafu. Pseudokód této fáze algoritmu je znázorněn v ukázce 5. Algoritmus přebírá na vstupu původní graf a graf podobností a jako výstup vrací strukturu obsahující komprimovanou podobu původního grafu $C = (N, Ex, Ri, Rl)$ kde N označuje množinu vrcholů grafu, Ex je uspořádaná dvojice $(n, M) : n \in N; M \subset N$, kde M je *uspořádaná* množina sousedů vrcholu n, které nebyly namapovány na žádného ze sousedů odkazovaného vrcholu (extra vrcholy), Ri je uspořádaná dvojice $(n_1, n_2) : n_1 \in N; n_2 \in N$, vyjadřující vztah, kde vrchol n_1 odkazuje na vrchol n_2 a Rl je uspořádaná dvojice $(n, L) : n \in N; \forall l \in L : l \in \{0, 1\}$, kde L je uspořádaná n-tice reprezentující referenční seznam.

Při vytváření komprimovaného grafu je důležité, abychom pro uložení referenčního seznamu vybrali vhodnou datovou strukturu. V případě, kdy bychom zvolili například pole celých čísel, byl by algoritmus bezúčelný, protože bychom zachovali stejné množství paměti pro reprezentaci hrany mezi dvěma vrcholy. Každá hodnota v referenčním listu je prakticky binární a proto nám pro její reprezentaci stačí pouze 1 bit. Hodnota 1 vyjadřuje existenci hrany mezi původním vrcholem a vr-

cholem umístěným na odpovídající pozici v uspořádané množině extra vrcholů odkazovaného vrcholu a hodnota 0 naopak reprezentuje neexistenci této hrany. V případě této práce byla pro reprezentaci referenčního listu použita třída *BitArray*[14], představující kompaktní strukturu udržující pole bitů.

Algoritmus 5: CreateCompressedGraph

Data: $G = (V, E)$ původní graf; $G' = (V', E')$ graf podobností
Result: $C = (N, Ex, Ri, Rl)$

```
1  begin
2      for x ∈ V' do
3          if V'_{G'}(x) = ∅ then
4              N_C(x)_Ex ⟵ V_G(x)
5          else
6              N_C(x)_Ri ⟵ {x}
7              y ⟵ V'_{G'}(x) //z povahy věci obsahuje vždy max jednoho souseda
8              N_C(x)_Ex ⟵ V_G(x) − (V_G(x) ∪ V_G(y))
9              for z ∈ V_G(y) do
10                 // množina sousedů V_G(y) musí být uspořádaná
11                 if z ∩ V_G(x) = ∅ then
12                     N_C(x)_Rl ⟵ N_C(x)_Rl ∪ {0}
13                 else
14                     N_C(x)_Rl ⟵ N_C(x)_Rl ∪ {1}
15                 end
16             end
17         end
18     end
19     return C
20 end
```

4.4 Popis algoritmu dekomprese

Kromě komprimace grafu je také nutné implementovat způsob, jakým lze graf dekomprimovat. Jak již zde bylo několikrát zmíněno, tak hlavní podmínkou tvorby kompresního algoritmu bylo to, aby při dekompresi nebylo nutné dekomprimovat celý graf, ale pouze tu část grafu, se kterou chceme pracovat. S ohledem na tuto podmínku byl implementován rekurzivní algoritmus, který je znázorněn v ukázce 6.

Vstupem algoritmu je výchozí vrchol x a komprimovaný graf C. Výstupem algoritmu je pak uspořádané pole sousedů daného vrcholu x. V případě potřeby prohledávání grafu například metodou DFS nebo BFS, je možné využít tento algoritmus pro dekompresi jednotlivých vrcholů v průběhu prohledávání grafu. Je tedy jasné, že v případě práce s komprimovaným grafem dochází ke snížení rychlosti průchodů grafem oproti původní reprezentaci grafu.

Algoritmus 6: DecompressNode

Data: $C = (N, Ex, Ri, Rl)$ komprimovaný graf; x výchozí vrchol
Result: uspořádané pole sousedů vrcholu x grafu C

```
1  begin
2      if N_C(x)_Ri = ∅ then
3          return N_C(x)_Ex
4      end
5      referenceNode ⟵ N_C(x)_Ri //z povahy věci obsahuje množina 1 prvek
6      refNodeNeighbors ⟵ DecompressNode(C, referenceNode)
7      curNodeNeighbors ⟵ []
8      for i ∈ 0..|N_C(x)_Rl| do
9          if N_C(x)_Rl_i = 1 then
10             curNodeNeighbors ⟵ curNodeNeighbors ∪ refNodeNeighbors[i]
11         end
12     end
13     curNodeNeighbors ⟵ curNodeNeighbors ∪ N_C(x)_Ex
14     Sort(curNodeNeighbor)
15     return curNodeNeighbors
16 end
```

Je také důležité si všimnout důležitosti uspořádání při reprezentaci sousedů daného vrcholu v komprimovaném grafu. Referenční seznam je realizován jako uspořádaná k-tice, kde každý prvek může nabývat hodnoty pouze 0 nebo 1 a pozice určuje vztah k danému sousedovi. Pokud by množina extra vrcholů $N_C(x)_{Ex}$ pro vrchol x v komprimovaném grafu C nebyla uspořádaná, bylo by nutné realizovat referenční seznam jiným a paměťově nákladnějším způsobem.

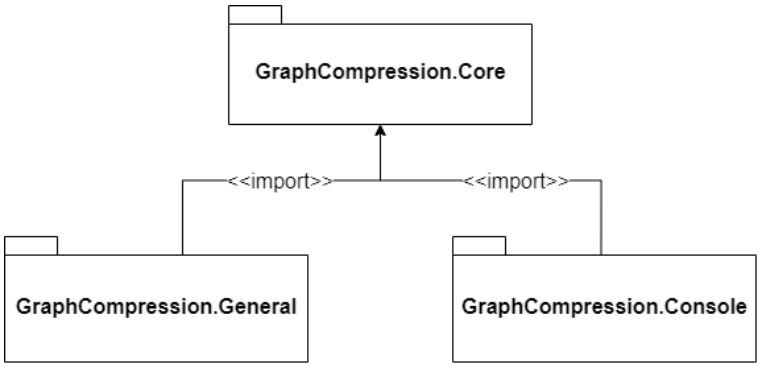
V souvislosti s implementací této metody bychom se mohli ptát, jak velkou část grafu je tedy nutné pro získání jednoho vrcholu dekomprimovat. Tuto otázku řeší parametr *hloubka referenčního zřetězení*, který je popsán v kapitole 5.

4.5 Technické detaily implementace

Pro finální implementaci algoritmu byl zvolen programovací jazyk C# ve verzi 9.0 spolu s .NET frameworkem ve verzi .NET 5.0. Celý projekt byl rozdělen do tří samostatných komponent (projektů), jejichž struktura je znázorněna na obrázku 4.3.

4.5.1 GraphCompression.Console

Komponenta *GraphCompression.Console* je jednoduchá konzolová aplikace, která byla v průběhu vývoje využívaná pro testování a následně i pro spouštění experimentů a jejich následné vyhodnocení. Po dokončení implementace byla komponenta upravena tak, aby obsahovala jednoduchou ukázku použití kompresního algoritmu včetně ukázky načtení dat ze souboru a vytvoření základní



Obrázek 4.3: Struktura projektu

struktury grafu s pomocí komponenty *GraphCompression.General*. Kromě načtení a komprese grafu je v ukázce znázorněn způsob, jakým je možné provést serializaci komprimované struktury do binárního souboru včetně dekomprese a sestavení komprimovaného grafu z tohoto souboru.

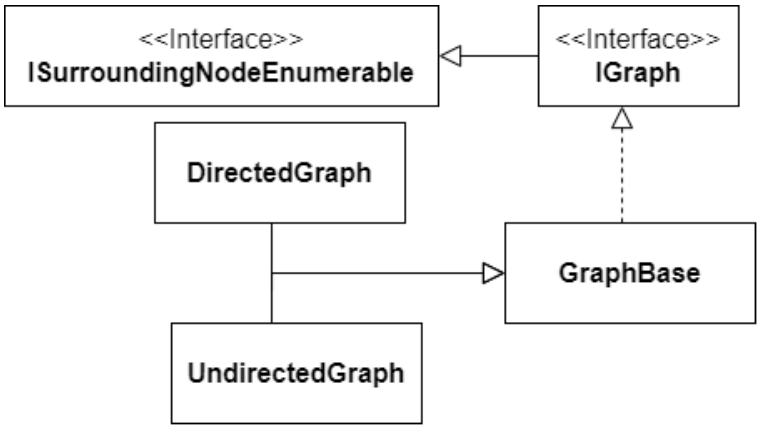
4.5.2 GraphCompression.General

Komponenta *GraphCompression.General* obsahuje implementaci metod umožňující načtení grafu z různých datových formátů a vytvoření základní struktury grafu reprezentované seznamem sousedů. Pro reprezentaci základní struktury grafu byl použit slovník (*Dictionary*) spolu s množinou (*HashSet*) pro reprezentaci sousedů. Tato datová struktura byla následně začleněna do tříd *DirectedGraph* a *UndirectedGraph*, které obsahují metody umožňující manipulaci s grafem. Na obrázku 4.4 je znázorněn třídní diagram zachycující statický pohled rozvržení tříd a rozhraní používaných pro základní reprezentaci grafu. Všechny tyto třídy jsou ve skutečnosti implementovány v komponentě *GraphCompression.Core*. Komponenta *GraphCompression.General* je tudíž na této komponentě závislá, jak znázorňuje obrázek 4.3.

Hlavním cílem vyčlenění logiky načítání dat a vytváření grafu byla snaha o oddělení logiky kompresního algoritmu od ostatních částí programu. Tímto rozdělením bylo docíleno snížení provázanosti jednotlivých částí algoritmu, což umožňuje použit případně i distribuovat komponentu *GraphCompression.Core* jako samostatnou knihovnu. K tomu přispívá i fakt, že je vstup pro kompresní algoritmus postaven na rozhraní *IGraph*, což umožňuje, využití kompresního algoritmu i pro jiné způsoby základní reprezentace grafu, než je seznam sousedů použitý v případě komponenty *GraphCompression.General*.

4.5.3 GraphCompression.Core

Komponenta *GraphCompression.Core* již obsahuje kompletní implementaci kompresního algoritmu, která je rozdělena do dvou oblastí, kdy jedna oblast kompletně pokrývá první fázi algoritmu popsa-



Obrázek 4.4: Struktura tříd pro základní reprezentace grafu

nou v kapitole 4.3.1 a druhá oblast pokrývá druhou a třetí fázi algoritmu popsanou v kapitole 4.3.2 a 4.3.3.

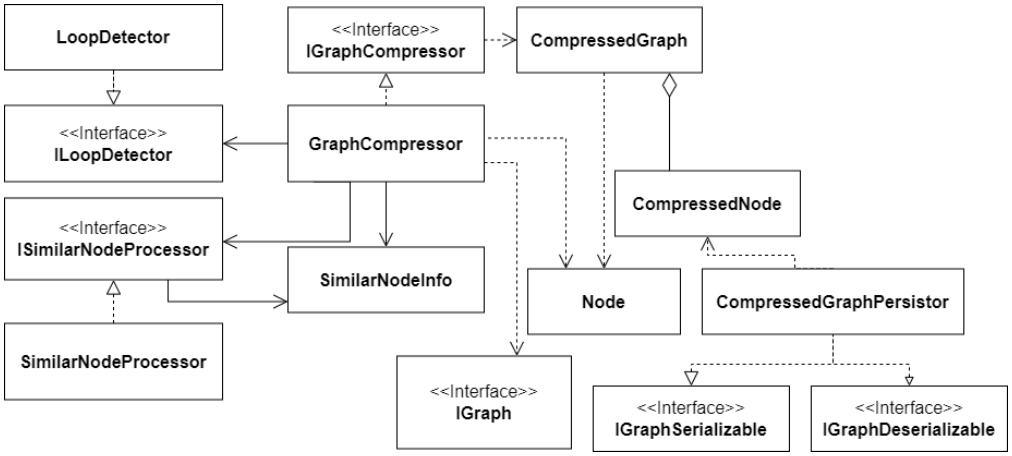
První fáze algoritmu (vytvoření grafu podobností) je umístěna v třídě *SimilarNodeProcessor*. Třída nabízí jednu veřejnou metodu *CreateSimilarityGraph* definovanou rozhraním *ISimilarNodeProcessor*, která na vstupu přebírá graf definovaný rozhraním *IGraph* a jako výstup vrací kolekci prvků *SimilarNodeInfo* reprezentující vztah podobnosti pro jednotlivé uzly grafu. Jako jeden z optimalizačních prvků první fáze algoritmu bylo zapojení logiky pro hledání pouze okolní vrcholu. Tato logika ovšem není součástí této třídy. Vzhledem k snaze o zachování nezávislosti na vstupní datové struktuře a tomu, že nalezení okolí vrcholů je specifické, jak pro jednotlivé způsoby reprezentace grafu, tak i pro jednotlivé typy grafů (orientovaný a neorientovaný), byla logika přesunuta do tříd *UndirectedGraph* a *DirectedGraph*. Z toho vyplývá, že v případě použití navrženého kompresního algoritmu s jinou datovou strukturou, než je implementovaná v komponentě *GraphCompression.General* je nutné implementovat i logiku pro nalezení okolí vrcholu tak, jak definuje rozhraní *IGraph* respektive *ISurroundingNodeEnumerable*.

Druhá a třetí fáze algoritmu je implementována v rámci tříd *GraphCompressor* a *LoopDetector*. Třída *GraphCompressor* mimo finální sestavení komprimovaného grafu specifikovanou třetí fází rovněž kontroluje celý průběh komprese a je tak jedinou třídou, kterou je možné z této komponenty použít. Třída obsahuje jednu veřejnou metodu *Compress* definovanou rozhraním *IGraphCompressor*. Tato metoda na vstupu přebírá graf definovaný rozhraním *IGraph* a výstupem metody je komprimovaný graf definovaný třídou *CompressedGraph*. Třída obsahuje závislosti na třídy definované rozhraními *ISimilarNodeProcessor* vykonávající první fázi kompresního algoritmu a *ILoopDetector* implementováno třídou *LoopDetector* a obsahující implementaci pro detekci cyklů v grafu podobnosti využívanou v rámci druhé fáze algoritmu.

Po zavolání metody *Compress* nad třídou *GraphCompressor* je jako první vykonána první fáze algoritmu. Po ukončení první fáze a získání kolekce prvků *SimilarNodeInfo* je z této kolekce vytvořen

graf podobností. Z optimalizačního hlediska je v průběhu vytváření grafu podobností využívána právě třída *LoopDetector*, která již v této fázi odstraňuje z grafu podobností cykly.

Jakmile je vytvořen graf podobností, je ukončená druhá fáze algoritmu a začíná třetí fáze algoritmu (finální sestaveni komprimovaného grafu). V této fázi již třída *GraphCompressor* pouze na základně grafu podobností sestaví finální strukturu grafu, definovanou třídou *CompressedGraph* tak, že pro každý vrchol sestaví referenční seznam, seznam extra vrcholů a vytvoří vazbu mezi daným vrcholem a vrcholem, který je mu nejpodobnější (v případě, že se podařilo takový vrchol najít). Na obrázku 4.5 je pak možné vidět pohled na strukturu tříd v komponentě *GraphCompression.Core*. V příloze B je pak znázorněna detailnější popis struktury nejdůležitějších tříd v rámci všech komponent projektu.



Obrázek 4.5: Struktura tříd komponenty GraphCompression.Core

Kapitola 5

Experimenty a jejich vyhodnocení

Po dokončení implementace kompresního algoritmu proběhla analýza, která měla za cíl určit silné a slabé stránky implementovaného algoritmu. V průběhu analýzy bylo zjištěno, že vzhledem k potřebě komplexnější struktury komprimovaného grafu nejsou všechny grafy pro použití v tomto algoritmu vhodné. V některých případech docházelo k tomu, že velikost výsledné reprezentace grafu přesáhla velikost původní reprezentace grafu. V rámci této kapitoly budou nastíněny některé problémy implementovaného algoritmu. V rámci analýzy byly použity jak reálné grafy získané z veřejně dostupných zdrojů [15] a [16], tak i umělé grafy vytvořené tak, aby reprezentovaly nějaké obecné vlastnosti grafů.

Poznámka 1 Pro lepší orientaci jsou v tabulkách před jménem označeny hvězdičkou ($\star$) ty grafy, pro které byla komprese neúspěšná.

Název	**Typ**	**Počet vrcholů**	**Počet hran**	**Hustota**	**Poznámka**
HR	Neorientovaný	54 573	498 202	0,00033	reálný
web-NotreDame	Orientovaný	325 729	1 497 134	0,00001	reálný
musae_PTBR	Neorientovaný	1 912	31 299	0,01713	reálný
musae_git	Neorientovaný	37 700	289 003	0,00041	reálný
bio-DM-HT*	Neorientovaný	5 956	9 009	0,00051	reálný
randomGraph1*	Neorientovaný	5 000	6 228	0,00050	umělý
randomGraph2	Neorientovaný	5 000	625 858	0,05008	umělý
randomGraph3	Neorientovaný	5 000	100 084	0,00800	umělý
socfb-MIT	Neorientovaný	6 402	251 230	0,01226	reálný
socfb-nips-ego*	Neorientovaný	2 888	2 981	0,00072	reálný
path*	Neorientovaný	100 000	9 999	0,00000	umělý
CompleteGraph	Neorientovaný	1 000	499 500	1,00000	umělý

Tabulka 5.1: Základní informace o analyzovaných grafech

V tabulkách 5.1 a 5.2 jsou popsány grafy, které byly uvažovány v rámci analýzy algoritmu spolu s jejich základními vlastnostmi. Tabulka 5.1 obsahuje informace o typu grafu počtu vrcholů a hran spolu s hustotou a informací o zdroji grafu. Tabulky 5.2 pak obsahuje strukturální vlastnosti grafů jako je minimální, maximální a průměrný stupeň vrcholů, počet komponent souvislostí, modularitu a průměrný shlukovací koeficient. Chování algoritmu pak bude v rámci této kapitoly posuzováno oproti většině z těchto parametrů.

Pro testování algoritmu byla vytvořena samostatná komponenta, která prováděla automatické testy všech grafů definované v konfiguračním souboru. Komponenta provedla komprimaci grafu a následně byly výsledky uloženy do souboru ve formě JSON.

Komponenta v průběhu komprese grafu zaznamenávala čtyři parametry *Čas komprese*, *Kompresní poměr*, *Počet detekovaných referencí* a *Počet detekovaných cyklů*. V tabulce 5.3 jsou pak zaznamenány pro každý zmíněný graf výsledné hodnoty analýzy komprese.

Kompresní poměr udává rozdíl mezi původní velikosti grafu a velikostí grafu po kompresi. Pro jeho získání je tedy nutné znát obě tyto velikost. Velikost původního grafu je označena jako S_o a pro jeho výpočet je použit následující vzorec

$$S_o = 64 \left[2|V(G)| + \sum_{i=0}^{|V(G)|} deg(i) \right]$$

Vzhledem k tomu, že každý vrchol grafu je reprezentován celým číslem, je ve vzorci použita konstanta 64 reprezentující počet bitů potřebných pro uložení identifikátoru jednoho vrcholu. Celý graf je v původní podobě reprezentován pomocí *seznamu sousedů*. Z tohoto důvodu se velikost původního grafu spočítá jako počet bitů potřebných pro reprezentaci každého jednoho vrcholu plus počet potřebných bitů pro reprezentaci každého souseda pro daný vrchol, přičemž počet vrcholů je ve vzorci zahrnut dvakrát z důvodu nutné reference na seznam sousedů pro kterou rovněž počítáme teoretickou velikost 64 bitů.

Velikost komprimovaného grafu je pak označena jako S_c a vypočítá se pomocí následujícího vzorce přičemž bereme v úvahu strukturu komprimovaného grafu $C = (N, Ex, Ri, Rl)$ popsanou v kapitole 4.3.3.

$$S_c = \sum_{v=0}^{|N|} \Big[64 + 64|N_C(v)_{Ri}| + |N_C(v)_{Rl}| + 64|N_C(v)_{E_x}| \Big]$$

Stejně jako v případě původního grafu, tak i zde počítáme jako teoretickou velikost reference a reprezentace celého čísla 64 bitů. Velikost komprimovaného grafu je pak spočítána jako součet počtu bitů potřebných pro reprezentaci identifikátoru vrcholu, identifikátoru odkazovaného vrcholu, referenčního seznamu a velikosti seznamu extra vrcholů.

Název	**Minimální stupeň**	**Maximální stupeň**	**Průměrný stupeň**	**Počet komponent souvislosti**	**Modularita**	**PSK**
HR	2	420	18	1	0,737	0,143
web-NotreDame	1	10 721	5	1	0,935	0,235
musae_PTBR	1	767	33	1	0,287	0,340
musae_git	1	9 458	15	2	0,450	0,193
bio-DM-HT*	1	56	3	26	0,813	0,058
randomGraph1*	1	10	2	488	0,719	0,000
randomGraph2	194	321	250	1	0,053	0,050
randomGraph3	40	63	20	1	0,142	0,008
socfb-MIT	1	708	78	1	0,391	0,284
socfb-nips-ego*	1	769	2	1	0,809	0,803
path*	1	2	2	1	0,979	0,000
CompleteGraph	1	999	999	1	0,000	1,000

Tabulka 5.2: Strukturální vlastnosti grafů v analýze

Název	**Čas komprese [s]**	**Kompresní poměr**	**Úspora [%]**	**Počet referencí**	**Počet cyklů**
HR	05,60	0,7974	20,26	29 749	2 334
web-NotreDame	47,94	0,6853	31,47	60 375	7 055
musae_PTBR	00,41	0,5944	40,56	1 525	25
musae_git	13,51	0,8750	12,50	15 302	385
bio-DM-HT*	00,11	1,1395	-13,95	333	104
randomGraph1*	00,19	1,3507	-35,07	0	0
randomGraph2	29,95	0,9329	06,71	4 140	860
randomGraph3	02,80	0,9916	00,84	2 705	964
socfb-MIT	03,65	0,6119	38,81	5 196	394
socfb-nips-ego*	00,10	1,4545	-41,45	4	2
path*	00,42	1,4769	-47,69	0	0
CompleteGraph	03,70	0,0196	98,04	999	1

Tabulka 5.3: Výsledky analýzy algoritmu

Kompresní poměr je pak označen jako KR a je jednoduše vyjádřen jako podíl velikosti komprimovaného grafu k velikosti původního grafu podle následujícího vzorce

$$KR = \frac{S_c}{S_o}.$$

Pokud je výsledek kompresního poměru menší než 1 znamená to, že byla komprese úspěšná, a že výsledná struktura potřebuje pro reprezentaci grafu menší počet bitů. Naopak, pokud je výsledek kompresního poměru větší než 1, pak výsledná struktura potřebuje pro reprezentaci původního grafu větší množství paměti.

Pro jednodušší vyjádření je pak možné zavést ještě parametr *úspora* označován jako D, který vyjadřuje procentuální poměr uspořené paměti, a kterou získáme pomocí následujícího vzorce

$$D = 100(1 - KR).$$

Pro tento parametr platí, že kladné číslo vyjadřuje úsporu, zatímco záporné číslo vyjadřuje narůst paměti. Už na první pohled je z tabulky 5.3 jasné, že hlavní roli v úspěšné kompresi bude hrát roli první fáze algoritmu, ve které je vytvářen graf podobností. Počet odkazů je pak hlavním ukazatelem úspěšnosti této fáze. Pokud se v první fázi podaří najít k dostatečnému množství vrcholů jiný nejpodobnější vrchol, pak je to předpokladem k tomu, že bude komprese úspěšná a velikost výsledné reprezentace grafu bude menší než velikost původní reprezentace. Obecně nelze stanovit hranici, při které by se úspěšnost algoritmu překlopila, a to z toho důvodu, že úspěšnost komprese závisí i na jiných faktorech. Tento fakt můžeme pozorovat i v tabulce 5.4, která obsahuje procentuální vyjádření toho, kolik vrcholů v komprimovaném grafu obsahuje odkaz na jiný vrchol.

Název	**Počet vrcholů**	**Počet odkazů**	**Množství namapovaných vrcholů** [%]
HR	54 573	29 749	54,51
web-NotreDame	325 729	60 375	18,54
musae_PTBR	1 912	1 525	79,76
musae_git	37 700	15 302	40,59
bio-DM-HT*	5 956	333	5,59
randomGraph1*	5 000	0	0,00
randomGraph2	5 000	4 140	82,80
randomGraph3	5 000	2 705	54,10
socfb-MIT	6 402	5 196	81,16
socfb-nips-ego*	2 888	4	0,14
path*	100 000	0	00,00
CompleteGraph	1 000	999	99,90

Tabulka 5.4: Procentuální vyjádření množství namapovaných vrcholů grafů

Můžeme například vidět, že i přesto, že v případě grafu *randomGraph3* se podařilo namapovat 54% vrcholů, tak i přesto se výsledná úspora pohybuje pouze okolo 1% a například u graf *web-NotreDame* se podařilo namapovat necelých 19% a celková úspora je necelých 32%. Samozřejmě je jasné, že pokud se v první fází nepodaří namapovat žádný vrchol, jako je tomu v případě grafu *randomGraph1* a *path* pak bude komprese vždy neúspěšná. Poměr namapovaných vrcholů se odvíjí především od vnitřní struktury grafu a definovaných omezení kladených na algoritmus popsaných v kapitole 4.3. V tabulce 5.3 je také zaznamenáno pro každý graf počet detekovaných cyklů. V kapitole 4.3 bylo řečeno, že pro správné fungování komprese je nutné se zbavit všech cyklů v grafu podobností. Zbavením bylo myšleno odstranění vazby mezi vrcholy tvořící cyklus, čímž bylo docíleno určitého snížení počtu namapovaných vrcholů. Jak je ovšem vidět z tabulek 5.1 a 5.3, tak počet detekovaných cyklů je v poměru k počtu vrcholů ve většině případů zanedbatelný, a proto je možné říct, že fáze odstranění cyklů má pouze minimální vliv na výsledek komprese.

Podíváme-li se na ostatní parametry analyzovaných grafů, tak zjistíme, že neexistuje žádné pravidlo, pomoci kterého bychom mohli dopředu říct, který graf se podaří úspěšně komprimovat, a který naopak není pro tento algoritmus vhodný. Parametr, který bychom mohli uvažovat jako nejvhodnějšího kandidáta pro predikci úspěšnosti komprese je *hustota* grafu. Z dostupných dat je ovšem možné pouze říct, že pokud je komprese neúspěšná, pak je hustota grafu nízká. Neznamená to však, že nízká hustota grafu znamená neúspěch komprese. O tom je možné se přesvědčit v případě grafů *web-NotreDame* nebo *HR*, které mají poměrně malou hustotu, nicméně poměrně velkou úsporu po provedení komprese.

Jako další parametr grafu, který je dobré uvažovat v souvislosti s algoritmem je *průměrný stupeň*. V tomto případě bychom mohli na základě popisu principu algoritmu předpokládat, že nízký stupeň bude značit i nízkou úroveň komprese. Je to dáno zejména tím, že v algoritmu záměrně filtrujeme vrcholy s nízkým stupněm kvůli vyšším nárokům na reprezentaci komprimovaného grafu. Tuto domněnku je možné si potvrdit i na základě dostupných dat. Vidíme, že všechny grafy, které při kompresi selhaly mají velmi malý průměrný stupeň. V důsledku to potom znamená, že většina vrcholů má velmi malý počet sousedů a z tohoto důvodu se v první fází algoritmu do výpočtu vůbec nedostanou.

Pokud bychom se pak podívali na hodnoty zbylých parametrů, tak bychom zjistili, že neexistuje přímá souvislost mezi hodnotami parametru a výsledkem komprese. Toto chování můžeme pozorovat například u parametru průměrného shlukovacího koeficientu, kdy vidíme, že graf *socfb-nips-ego* má vysoký shlukovací koeficient (0,803) a graf *path* má nízký shlukovací koeficient (0) a i přesto byla komprese obou grafů neúspěšná.

Podíváme-li se do tabulky 5.1, zjistíme, že poslední dva zmíněné grafy *path* a *CompleteGraph* jsou umělé. Tyto 2 grafy byly generovány s úmyslem poukázat na důležitost vnitřní struktury grafu pro výsledek komprese. Graf *path* je generován tak, aby vytvořil cestu. Každý vrchol v tomto grafu (kromě prvního a posledního) má tedy vždy 2 sousedy (předchůdce a následníka). Oproti tomu *CompleteGrah*, jak už z názvu vypovídá je kompletním grafem. Každý vrchol v tomto grafu je tedy

spojen se všemi ostatními vrcholy. Z tabulky 5.3 je pak možné zjistit, že graf *CompleteGraph* dosáhl úspory 98.04%, zatímco u grafu *path* naopak k původní velikost přibylo 47,69%.

Je jasné, že v kompletním grafu má každý z vrcholu velký stupeň a navíc každý vrchol má stejné sousedy jako všechny ostatní vrcholy. V rámci první fáze algoritmu je tedy s velmi velkou pravděpodobností pro každý z vrcholu nalezen soused, který mu je podobný a navíc tento soused obsahuje všechny sousedy původního vrcholu (kromě sama sebe). Využití referenčního listu bude tedy maximální a velikost bloku extra vrcholů naopak minimální.

Naopak v případě grafu obsahující pouze cestu je každý vrchol téměř jedinečný. Každý vrchol kromě prvního a posledního, které mají buď pouze předchůdce nebo následníka, má vždy stejného souseda pouze právě s předchůdcem a následníkem. V první fázi algoritmu tedy pro žádný z vrcholu není nalezen jiný podobný vrchol, protože nastavené parametry algoritmu neumožňují vytvořit referenci na jiný vrchol v případě, že neobsahuje dostatečný počet stejných sousedů. Vzhledem k tomu, že počet stejných sousedů bude v nejlepším případě jeden, tak se žádný z vrcholu nespáruje s žádným jiným vrcholem a tudíž je výsledný graf podobnosti prázdný.

V úvodu práce bylo řečeno, že se algoritmus, ze kterého bylo vycházeno, využívá především pro komprimaci webových grafů. Vzhledem k vlastnosti mocninného rozdělení stupňů vrcholů těchto grafů byla provedena analýza vztahu mezi distribucí stupňů a dosaženým kompresním poměrem. Předpoklad byl takový, že grafy s mocninným rozdělením vrcholů budou vykazovat lepší chování z hlediska výsledného kompresního poměru. Tento předpoklad nebyl prokázán. V příloze A jsou zobrazeny distribuce stupňů jednotlivých testovaných grafů. Podíváme-li se na tyto distribuce, zjistíme, že distribuce stupňů grafu *bio-DM-HT* víceméně odpovídá mocninnému rozdělení a přitom výsledný kompresní poměr vykazuje zvýšení množství paměti potřebné pro reprezentaci grafu. Naopak distribuce stupňů grafu *randomGraph2* odpovídá binomickému rozdělení a přitom výsledný kompresní poměr grafu vykazuje snížení množství spotřebované paměti.

5.1 Hloubka referenčního zřetězení

Na začátku bylo definováno, že hlavním cílem práce je vytvořit takový algoritmus, který umožní komprimovat graf tak, abychom zachovali možnost procházení grafem bez nutnosti dekomprese celého grafu. Mohli bychom se pak ptát, jak velkou část grafu je nutné dekomprimovat, abychom získali kompletní informaci o jednom vrcholu. Na tuto otázku pak odpovídá právě parametr *hloubka referenčního zřetězení*, který pro každý vrchol udává počet dalších vrcholů, které je nutné dekomprimovat spolu s tímto vrcholem.

Pokud by se pak hodnota hloubky referenčního zřetězení nějakého vrcholu blížila celkovému počtu vrcholů, znamenalo by to, že pro dekompresi tohoto vrcholu je nutné dekomprimovat téměř celý zbytek grafu. Tento problém by mohl nastat právě v případě grafu *path*, pokud by nastala situace, kdy by každý z vrcholů vyhodnotil jako nejpodobnější vrchol následovníka a my bychom pak

chtěli dekomprimovat první vrchol. I z tohoto důvodu byly do algoritmu začleněny určité omezující parametry.

V rámci této části práce byla provedena i analýza hloubky referenčního zřetězení. Výsledek analýzy je možné vidět na obrázcích 5.1 a 5.2. Ty obsahují pro každý testovaný graf (kromě grafu *randomGraph1* a *path*, které jsem pro tuto analýzu bezvýznamné) distribuci velikosti hloubky referenčního zřetězení.

Z distribucí jednotlivých hloubek referenčního zřetězení je patrné, že ve většině případů maximální hloubka referenčního zřetězení nepřesáhla hodnotu 10. Dále je také možné vidět, že u většiny vrcholů má hloubka zanoření tendenci se pohybovat spíše v nižších hodnotách a pouze minoritní část vrcholů má hloubku zanoření větší.

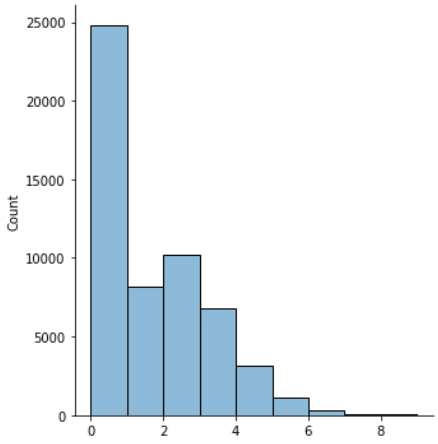
Největší hloubku zanoření má graf *web-NotreDame*. Opět je ale možné vidět, že téměř u všech vrcholů hloubka zanoření nepřesáhne hodnotu 5. Je tedy možné říct, že v 99% případů budeme muset dekomprimovat nanejvýš 5 dalších vrcholů grafu. V nejhorším případě pak bude nutné dekomprimovat 40 dalších vrcholu. Tato hodnota se může zdát vysoká, nicméně pokud vezmeme v úvahu počet vrcholů grafu, pak zjistíme, že i v nejhorším případě budeme muset dekomprimovat pouze 0,012% z celého grafu.

Z těchto údajů je patrné, že z časového hlediska je režie dekomprese vzniklá nutnosti dekomprimovat okolí daného vrcholu prakticky zanedbatelná. O tom je možné se přesvědčit v tabulce 5.5, která obsahuje informace o průměrném a maximálním čase dekomprimace vrcholu pro jednotlivé grafy.

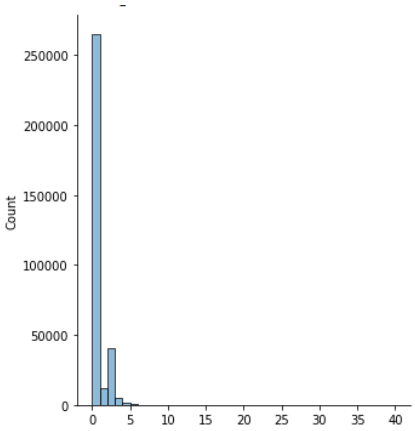
Název	**Průměrný čas [ms]**	**Maximální čas [ms]**
HR	1,8557	205
web-NotreDame	6,2470	395
musae_PTBR	0,6036	6
musae_git	0,8191	71
bio-DM-HT*	0	0
randomGraph1*	0,0011	5
randomGraph2	0,0356	18
randomGraph3	0,0004	1
socfb-MIT	0,4016	9
socfb-nips-ego*	0,0003	1
path*	0	0
CompleteGraph	21,3100	103

Tabulka 5.5: Časy dekomprese vrcholu jednotlivých grafů

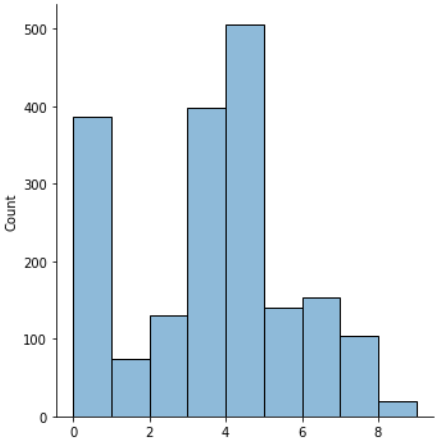
Z tabulky 5.5 je patrné, že jistá režie při dekompresi vzniká. Pokud vezmeme v úvahu graf *web-NotreDame*, který obsahuje 325 729 vrcholu a budeme vycházet z průměrného času potřebného pro dekompresi jednoho vrcholu, pak zjistíme, že průchod celým grafem bude vyžadovat přibližně 33 minut. Tento čas je samozřejmě pouze teoretický a ve skutečnosti se může lišit.



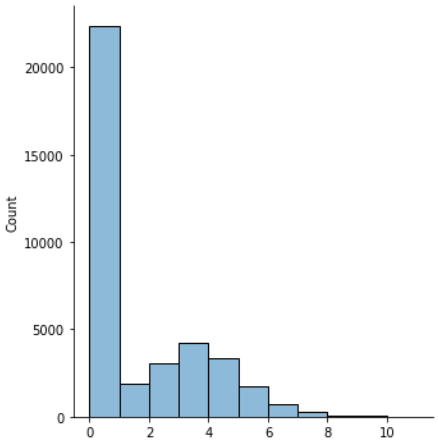
(a) HR



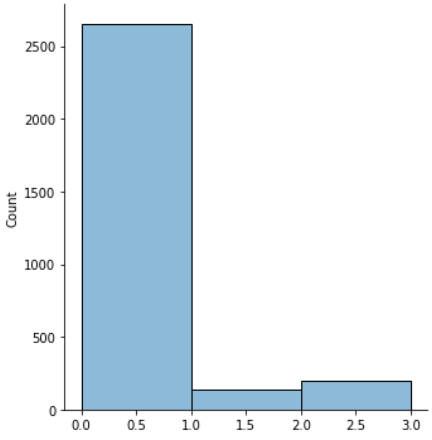
(b) web-NotreDame



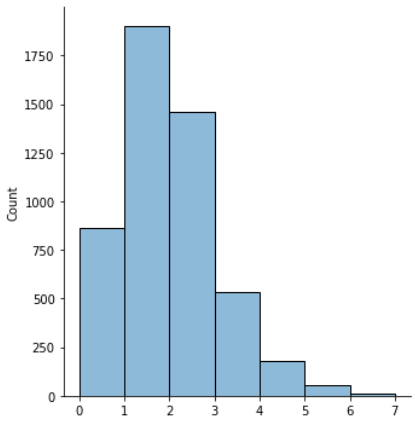
(c) musea_PTBR



(d) musea_git

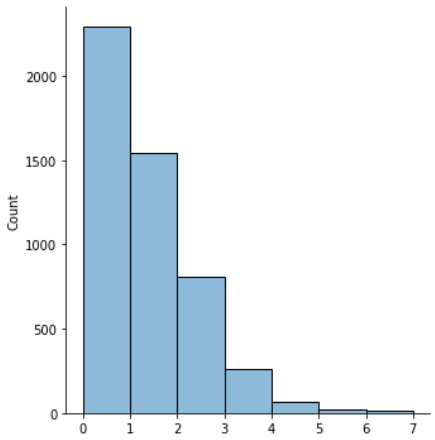


(e) bio_DM_HT

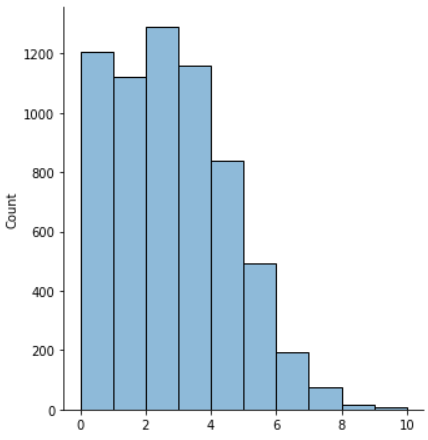


(f) randomGraph2

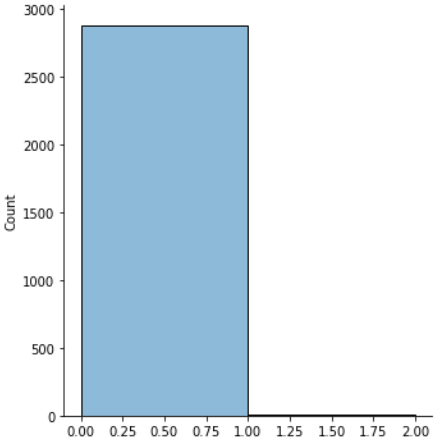
Obrázek 5.1: Distribuce hloubky referenčního zřetězení



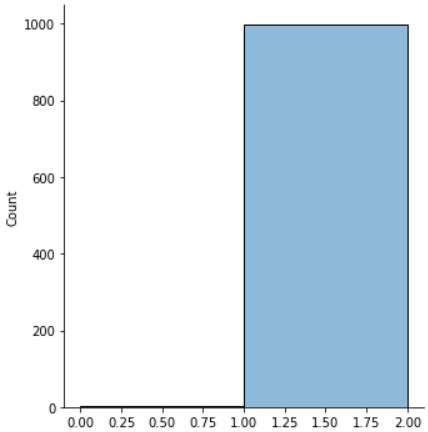
(a) randomGraph3



(b) socfb_MIT



(c) socfb_nips_ego



(d) CompleteGraph

Obrázek 5.2: Distribuce hloubky referenčního zřetězení

Kapitola 6

Závěr

Komprese obecně je velmi rozsáhlý problém. V mnoha odvětvích se na klade velký důraz na to, aby byly objemy dat co možná nejmenší případně pak na to, aby se data ukládala ve formě, která má co nejmenší nároky na využití paměti. V případě komprese grafu tomu není jinak i přesto, že díky stále se zvyšující velikost paměti počítačů není tento problém tak palčivý, jako v jiných odvětvích informatiky. Vývojáři při práci s grafy často neřeší množství využité paměti jak z důvodu, že mají pro práci dost pamětí, tak i z důvodu, že nepracují s tak rozsáhlými grafy. Tento přístup je také umocněn tím, že obecně algoritmy pro práci s grafy jsou časově velmi náročné a tudíž na velmi velké grafy prakticky neaplikovatelné. Z tohoto důvodu se často hledají alternativy, abychom se práci s velmi rozsáhlými grafy vyhnuli. Problém práce s velmi rozsáhlými grafy tak zůstává velkým korporacím, jako je Google, Facebook a pod.

Velkou otázkou však zůstává použitelnost komprimovaných struktur grafů. Jak již zde bylo naznačeno, tak algoritmy pracující s grafy mají ve většině případů poměrně velkou časovou složitost. Když k této časové složitosti připočteme zvýšený čas nutný k dekompresi komprimované reprezentace grafu, můžeme v mnoha případech narazit na problémy rychlosti výpočtu. Obecně pak platí, že zvýšením paměťové složitosti snížíme složitost časovou a naopak. Často se tak stává, že namísto toho, aby se uvažovalo o komprimované reprezentaci grafu se uvažuje o rozšířené reprezentaci, která naopak vyžaduje více paměti ale urychluje zpracování použitých algoritmů.

I přesto, že momentálně není na kompresi grafu kladen velký důraz, tak do budoucna je tato otázka určitě na místě. S rozmachem sociálních sítí začal objem dat reprezentovaný grafem prudce narůstat. Podle statistik registruje sociální síť Facebook k roku 2022 téměř 3 miliardy uživatelů. Paměťová náročnost reprezentace grafů s tak velkým počtem vrcholů je velmi vysoká a často na běžných zařízeních nerealizovatelná.

V rámci práce pak bylo v souvislosti s touto problematikou nastíněno několik možností komprese grafů. Současně s tím byla provedena implementace adaptace jedné z popsaných metod a následná analýza. Výsledkem analýzy bylo zjištění, že implementovaný algoritmus není vhodný pro všechny grafu a v některých případech může vykazovat dokonce narůst paměti potřebné pro reprezentaci

grafu. V opačných případech ovšem vykazoval poměrně velké snížení celkového množství paměti potřebné pro reprezentaci grafu. Analýza rovněž ukázala, že není možné předem stanovit, zda je graf pro implementovaný kompresní algoritmus vhodný či nikoliv, a to z toho důvodu, že výsledek komprese ve velké míře závisí na vnitřní struktuře grafu.

Literatura

1. SEO, Hojin; PARK, Kisung; HAN, Yongkoo; KIM, Hyunwook; UMAIR, Muhammad; KHAN, Kifayat Ullah; LEE, Young-Koo. An effective graph summarization and compression technique for a large-scaled graph. *The Journal of Supercomputing*. 2020, roč. 76, č. 10, s. 7906–7920. ISSN 1573-0484. Dostupné z DOI: 10.1007/s11227-018-2245-5.
2. KOVÁŘ, Petr. *Teorie grafu* [[online]]. 2022. Dostupné také z: https://homel.vsb.cz/~kov16/files/skriptum_teorie_grafu.pdf.
3. BOLDI, P.; VIGNA, S. The webgraph framework I. In: *Proceedings of the 13th conference on World Wide Web - WWW '04*. New York, New York, USA: ACM Press, 2004. ISBN 158113844X. Dostupné z DOI: 10.1145/988672.988752.
4. APOSTOLICO, Alberto; DROVANDI, Guido. Graph Compression by BFS. *Algorithms*. 2009, roč. 2, č. 3, s. 1031–1044. ISSN 1999-4893. Dostupné z DOI: 10.3390/a2031031.
5. BUEHRER, Gregory; CHELLAPILLA, Kumar. A scalable pattern mining approach to web graph compression with communities. In: *Proceedings of the international conference on Web search and web data mining - WSDM '08*. New York, New York, USA: ACM Press, 2008, s. 95–. ISBN 9781595939272. Dostupné z DOI: 10.1145/1341531.1341547.
6. MANETH, S.; PETERNEK, F. Compressing graphs by grammars. In: *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. 2016, s. 109–120. Dostupné z DOI: 10.1109/ICDE.2016.7498233.
7. SHAH, Rushabh. *Graph Compression Using Pattern Matching Techniques*. IEEE, 2018.
8. KHAN, Kifayat Ullah; NAWAZ, Waqas; LEE, Young-Koo. Lossless graph summarization using dense subgraphs discovery. In: *Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication - IMCOM '15*. New York, New York, USA: ACM Press, 2015, s. 1–7. ISBN 9781450333771. Dostupné z DOI: 10.1145/2701126.2701157.
9. ZHOU, Fang. Graph Compression. *Algorithm*. 2010.
10. LARSSON, N. J.; MOFFAT, A. Off-line dictionary-based compression. *Proceedings of the IEEE*. 2000, roč. 88, č. 11, s. 1722–1732. ISSN 1558-2256. Dostupné z DOI: 10.1109/5.892708.

11. WEISSTEIN, Eric W. *"NP-Problem." From MathWorld–A Wolfram Web Resource.* [B.r.]. Dostupné také z: https://mathworld.wolfram.com/NP-Problem.html.

12. *Voho / digitální wiki / informatika / algoritmus / grafový algoritmus.* [B.r.]. Dostupné také z: http://voho.eu/wiki/algoritmus-dfs/.

13. *Detect cycle in a directed graph.* 2021-11. Dostupné také z: https://www.geeksforgeeks.org/detect-cycle-in-a-graph/.

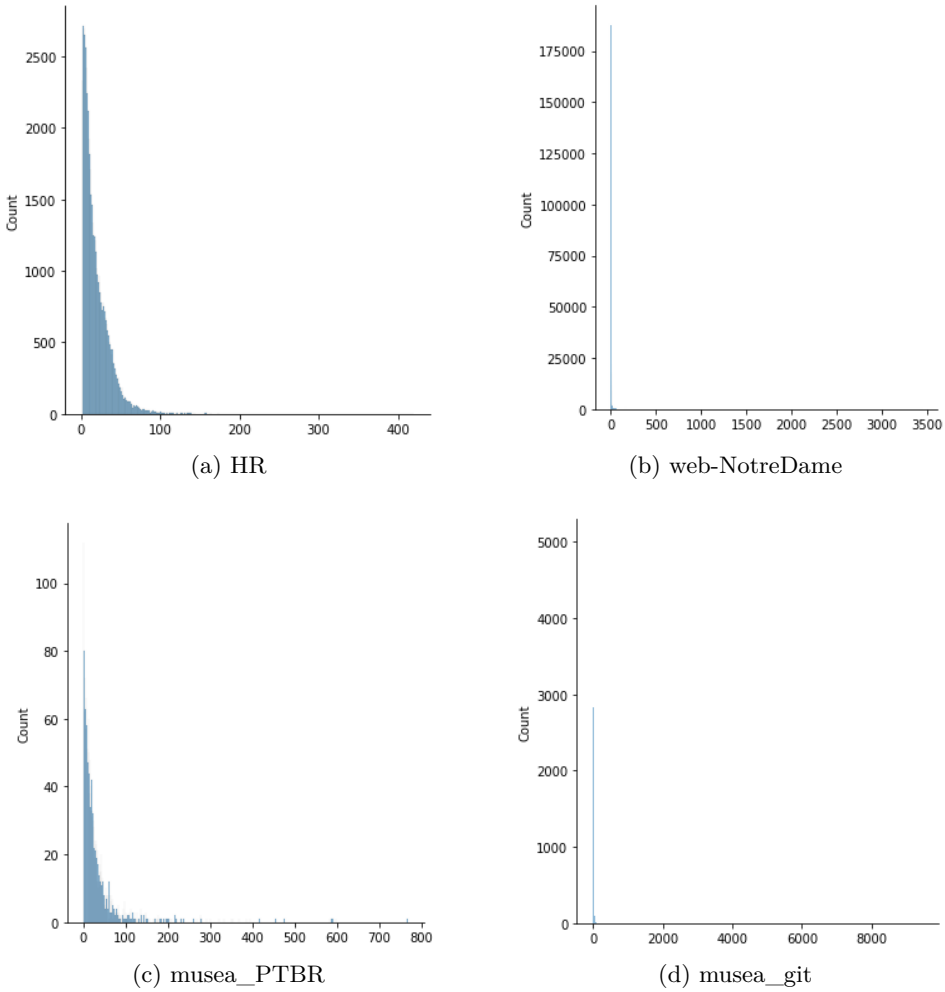
14. DOTNET-BOT. *Bitarray class (system.collections).* [B.r.]. Dostupné také z: https://docs.microsoft.com/en-us/dotnet/api/system.collections.bitarray?view=net-6.0.

15. ROSSI, Ryan A.; AHMED, Nesreen K. The Network Data Repository with Interactive Graph Analytics and Visualization. In: *AAAI*. 2015. Dostupné také z: https://networkrepository.com.

16. LESKOVEC, Jure; KREVL, Andrej. *SNAP Datasets: Stanford Large Network Dataset Collection* [http://snap.stanford.edu/data]. 2014-06.

Příloha A

Distribuce stupňů analyzovaných grafů



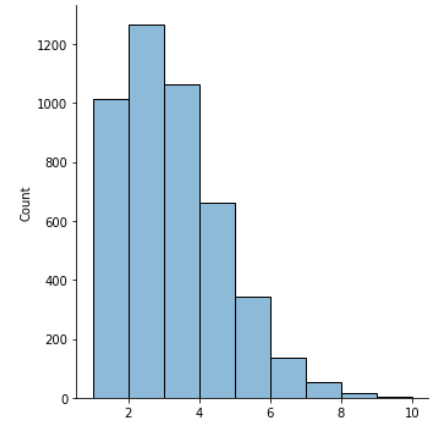
(a) HR

(b) web-NotreDame

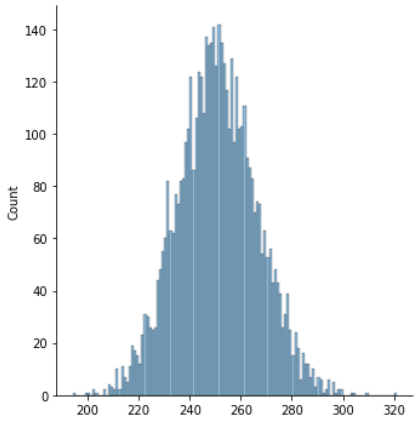
(c) musea_PTBR

(d) musea_git

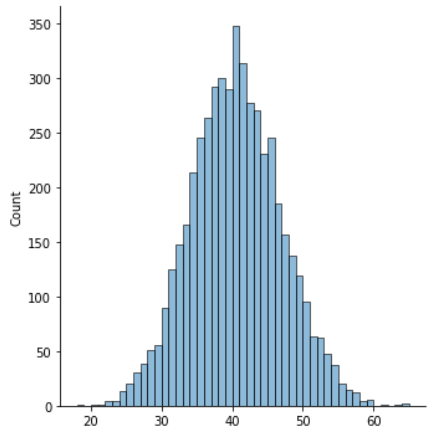
Obrázek A.1: Distribuce stupňů analyzovaných grafů



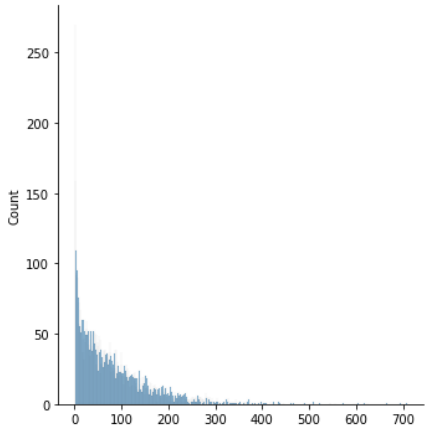
(a) randomGraph1



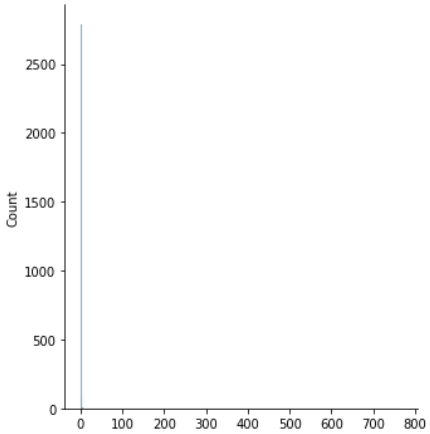
(b) randomGraph2



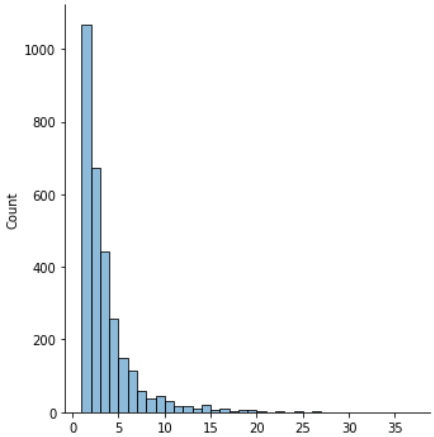
(c) randomGraph3



(d) socfb_MIT



(e) socfb_nips_ego

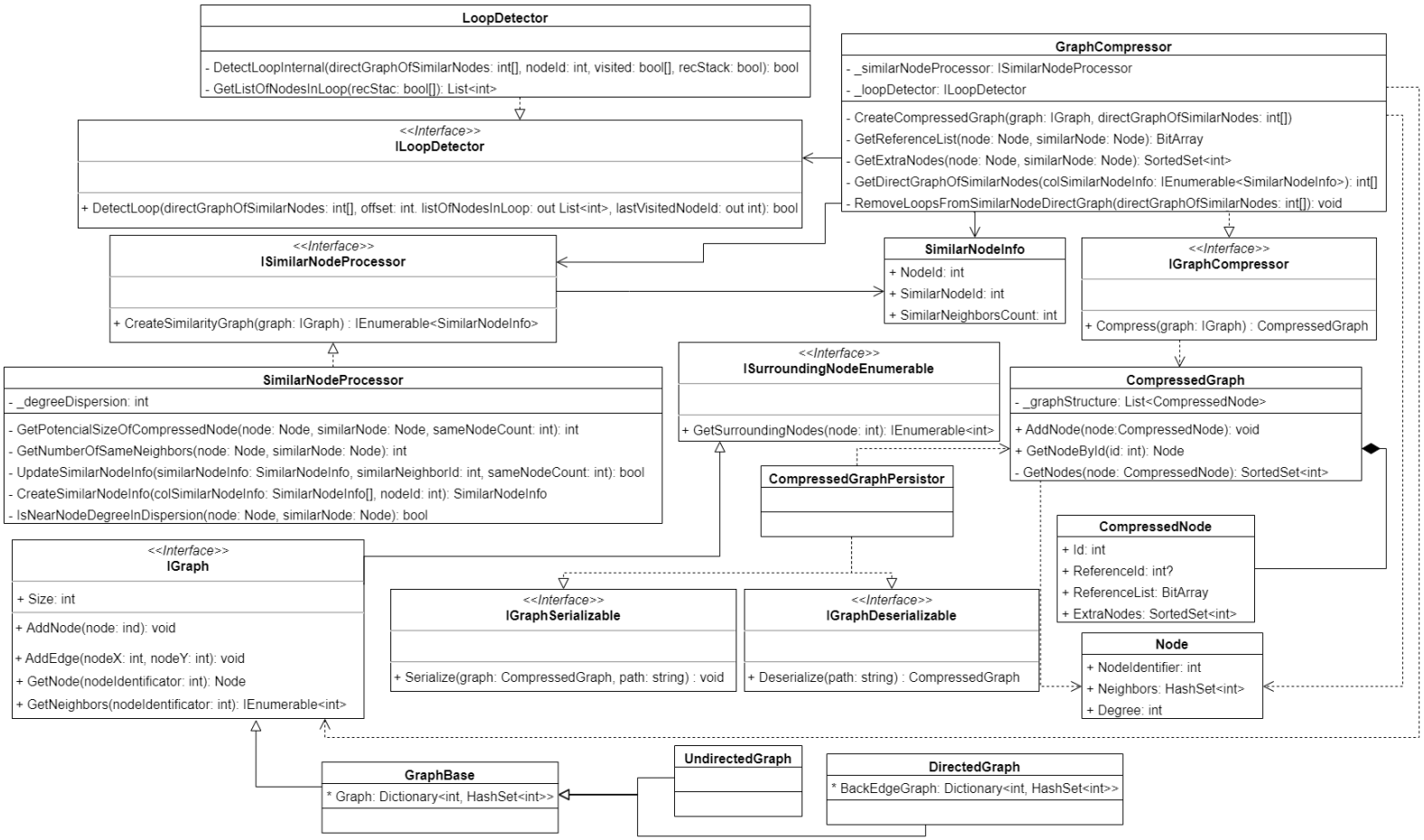


(f) bio_DM_HT

Obrázek A.2: Distribuce stupňů analyzovaných grafů

Příloha B

Struktura projektu



Obrázek B.1: Struktura tříd projektu